VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

3D Object Pose Estimation from RGB-D Data

Odhad vzájemné polohy objektu a kamery z RGB-D dat

2018 Petr Martínek

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Diploma Thesis Assignment

Student: **Bc. Petr Martínek**

Study Programme: N2647 Information and Communication Technology

Study Branch: 2612T025 Computer Science and Technology

Title: 3D Object Pose Estimation from RGB-D Data
Odhad vzájemné polohy objektu a kamery z RGB-D dat

The thesis language: English

Description:

The aim of the thesis is to implement an algorithm for estimating the mutual position of selected object and the camera from both depth and color data. The result will be a C++ console application that will allow to load the sought object represented by the point cloud and its sequential search in the sequence of test images. The matches found will be appropriately displayed (i.e. text output including transformation matrices of found poses, estimated error, and graphically rendered match of the found object with the test scene).

1. Learn about methods for estimation and tracking of mutual position of object and camera in video sequences.
2. Implement the selected depth based method with possible utilization of RGB channels in OpenCL or CUDA environment.
3. Demonstrate the functionality of the implemented method with appropriate test sequences (at least two models in non-trivial environments).
4. Evaluate the robustness of the method, especially with regard to object's size, symmetry and surface complexity.
5. Carefully document individual steps and procedures as well as achieved results.

References:

[1] Drost, B. et al.: Model globally, match locally: Efficient and robust 3D object recognition. In: CVPR. pp. 998–1005 IEEE Computer Society (2010).
[2] Choi, C., Christensen, H.I.: 3D pose estimation of daily objects using an RGB-D camera. In: IROS. pp. 3342–3349 IEEE (2012).
[3] Choi, C., Christensen, H.I.: RGB-D object pose estimation in unstructured environments. Robotics and Autonomous Systems. 75, 595–613 (2016).

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Tomáš Fabián, Ph.D.**

Date of issue: 01.09.2017

Date of submission: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
Head of Department

prof. Ing. Pavel Brandštetter, CSc.
Dean

I hereby declare that this master's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, April 30, 2018 ..... Petr Mokrnek

First and foremost, I would like to express my gratitude to my supervisor Ing. Tomáš Fabián, Ph.D., for giving me the opportunity to work on this fascinating thesis topic. His experience, understanding and patience helped me considerably during my work on this thesis and my master studies as a whole.

I would also like to thank my parents for their loving and caring support and for giving me the opportunity to earn my master degree. And finally I would like to thank my friends for being there for me.

Abstrakt

Estimace pozice a rotace (pózy) známého objektu ve scéně je jeden ze základních úkolů počítačového vidění. Má svoje využití u robotů zvedajících předměty v plně autonomních skladech nebo u rozšířené reality. Prezentujeme metodu pro estimaci pózy objektu, která využívá hloubková data a barvu scény, ale je schopná fungovat pouze s hloubkou. Je také schopná zvládat částečně zakryté objekty i snímačový šum. Výsledná multiplatformní C++ aplikace používá jak vlákna, tak OpenCL pro paralelizaci, a byla testována na synteticky vytvořených scénách, stejně jako na scénách zaznamenaných hloubkovými kamerami.

Klíčová slova: Odhad pózy, RGB-D, OpenCL, GPU, Paralelizace

Abstract

Estimation of position and rotation (pose) of a known object inside a scene is one of the fundamental tasks of computer vision. It has its use in robot picking in fully autonomous warehouses or augmented reality. We present a method for an object pose estimation, that utilizes scene depth and color information, but is able to function with depth alone. It is also capable of handling partial object occlusion and sensor noise. Resulting multi-platform C++ application uses both threads and OpenCL for parallelization, and was tested on synthetically generated scenes as well as scenes captured by depth cameras.

Key Words: Pose estimation, RGB-D, OpenCL, GPU, parallerization

Contents

List of symbols and abbreviations

AABB	–	Axis-Aligned Bounding Box
API	–	Application programming interface
CD	–	Compact Disc
CNN	–	Convolutional Neural Network
RGB-D	–	Red, Green, Blue and Depth
PPF	–	Point Pair Feature
SDK	–	Software Development Kit

List of Figures

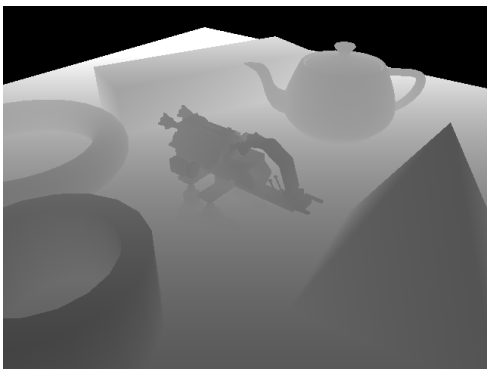
List of Tables

Listings

Introduction

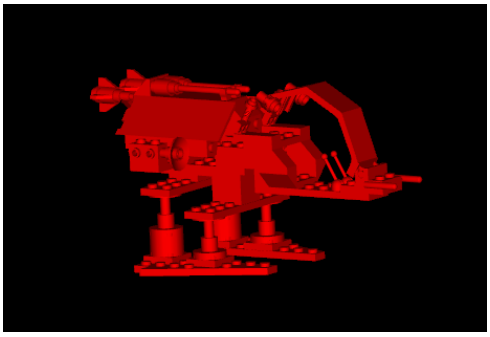
Pose estimation from RGB-D data refers to a process of determining the transformation of the specified 3D object in a 3D scene. It is one of the fundamental tasks of computer vision, essential in robot picking in fully autonomous warehouses or in augmented reality. This task is difficult due to high dimension of the problem (6 degrees of freedom), noise and missing values in the sensor data and occlusion of the objects.

This thesis will present a method that estimates the pose of an object represented by a point cloud of its model inside a scene described by the depth and color map. Figure 1 shows examples of these inputs. Our method should be robust enough to handle partial object occlusion and reasonable levels of noise, and missing values in the sensor data. We also expect, that color information may be unavailable on some of the depth sensors, or may not be usable, because of illumination changes. Therefore, our method should be fully functional with only the the depth data.



(a) Depth map

(b) Color map



(c) Sought model

Figure 1: Examples of the input data

We want our implementation to perform the pose estimation in real time as its usually required in robotics and almost always required in augmented reality. We expect to parallelize it on many-core architectures. Experiments will be conducted in order to test performance of this method. We will start with synthetically generated scenes as they are more suitable for

finding errors in our implementation. Once we are comfortable with results we will move on to experiments with real data obtained by the available sensors.

This thesis is organized into chapters. Below is a brief summary of each one.

State of the art This chapter covers the overview of the methods employed to solve the pose estimation problem and also mentions the evolutions of the hardware, that were necessary to achieve real time object pose estimation.

Model Globally, Match Locally This chapter describes the object pose estimation pipeline, that was first proposed in [10], and on which we based our implementation.

Implementation This chapter start with the description of the steps that need to be taken to transform the data from the sensors, which may contain noise and missing values, into data, that can be used by the method described in the previous section. This is then followed by the description of our implementation of the method described in the previous chapter. It also contains description of some additions and optimizations, that we have made in our implementation, along with the reasoning that let us to made those optimizations and additions. The last part of the chapter describes the parallelization on both multi-core and many-core architectures.

Experiments This chapter starts with the the description of the data, that was used during the experiments, and our methodology for the time and accuracy measurements. This is then followed by the results and evaluation of the experiments, that we have carried our to evaluate the following:

- Impact of our optimizations and additions on the pose estimation time and accuracy of the pose estimation.
- Many-core parallelization and scalability of multi-core parallelization.
- Actual usage with real data.

Conclusion Final chapter where we summarize achieved results, and propose which parts of our implementation may benefit from further research.

State of the art

Research in the field of object pose estimation in the 3D scenes obtained by various sensors, such as laser scanners, time-of-flight cameras and stereo systems, has been active for the last four decades. The addition of the third dimension distinguishes these methods from more traditional 2D matching methods, such as chamfer matching which works with the edges in the images. Whereas, methods based on depth data usually use 3D position and surface normals. Great number of them does not use edges. The ones that use edges do it usually only if they also have the the color information in addition to the depth one (RGB-D) because edges in the depth data are usually distorted.

Early works such as [4] (1985), [19] (1992), [9] (1997), [14] (1997), [6] (2001) exist. However, these works were very limited in their deployment because to the lack of computing power and unavailability of 3D capture devices. Publication [17] states that as of the year 2005 laser scanners were capable of high-speed acquisition of dense and accurate point clouds. The release of Kinect in 2010, its SDK for Windows in 2011, and its second edition in 2012 provided researches with very accessible and relatively low cost depth camera which was used in many works [16, 7, 3, 5, 12]. As of now Kinect is discontinued but many other low-cost depth cameras are on the market such as Orbbec Astra series or Intel RealSense.

Introduction of GPUs for general computations helped to solve the problem of performance. Method described in [11] managed to use GPU shaders, which are meant for realtime graphics and not general computing, perform pose estimation in the year 2007. Furthermore, it mentions usage general computing API for NVIDIA GPUs CUDA as possible future project (CUDA had its initial release in the year 2007). The follow up work [15] actually uses CUDA. GPU usage became the norm for the real-time methods of pose estimation [7, 3, 12, 5, 8], and most of the methods that use artificial neural networks.

Previously mentioned methods vary greatly and proper categorization is difficult. Authors of [10] (2010) characterize previously published methods as global [17, 11, 15] or local [19, 9, 14] based on the type of descriptors they use. They conclude that global methods are not very precise or fast unlike local ones. However, the latter methods requires dense information about the surface and have problems with symmetric objects and object occlusion. They are also more susceptible to the noise and missing values in the input data. They then propose a method thats is based on local features but creates a global model description from them which is then matched by a fast voting scheme. Global model description required only sparse set of oriented points which improves matching time. They declare that their implementation without any parallelization is faster than global methods using GPU and that it can handle object occlusion.

General usage of GPUs also led to new developments in the field of artificial neural networks. These networks have been very successful in solving computer vision problems. This lead to development of a great variety of new methods. For Example, authors of [18] decided to use convolutional neural network(CNN), which is very successful in other areas of computer vision.

Their implementation uses pre-trained CaffeNet [13] to extract the features from the RGB and Depth images, and then uses support vector machines to obtain object type, instance and pose. All inputs for their pre-trained network have to be 227×227 RGB images. Therefore, they have to use preprocessing to convert depth data into RGB images. The reason for the usage of pre-trained network was to avoid the long training period that also requires a large training dataset. Authors state that their method allows frame rates of up to 5 Hz.

CNN was also adopted in [20]. Authors call their version for pose estimation PoseCNN. PoseCNN performs three different tasks to estimate the pose of the object. First one is classification of each pixel in the input image into an object class. Second task is finding 2D center of an object and measuring the distance between the object and the camera in order to obtain translation of the object. Bounding boxes are also computed. Third task estimates the rotations of the object, which is combined with the translation from the previous task to obtain the object pose. Authors state that their method is able to function with vision data only.

More recent works that use other techniques than the usage of artificial neural network also exist. For example, [12] (2015), which estimates the pose of texture-less objects. This shows the variety of approaches the researches take. In the conclusion of the previously mentioned method, the authors state that their method can work with vision only data. Whereas, the aim of this method is the recognize objects, which are composed of one or more colors and have no other distinguishable informations such as pictures or text on them.

This method stars with a rough pose estimation with template matching and then runs a fine pose estimation which is defined as an optimization problem of finding the closest match between the object in the scene and the model reference and uses particle swarm optimization to obtain the solution.

Works such as [5] (2015) or [8] (2016) revise the pipeline used in the [10]. These two methods illustrate a variety of approaches the researches take. Even though they are both based on the same original research, [5] proposes a scene segmentation to aid the pose estimation while [8] avoids segmentation in order to make their method viable for highly cluttered scenes. Both of them use primary depth data and consider color only as an additional information because the method they base their research on works only with depth data and cannot be transformed into color only method.

Implementation in this thesis is also based on the pipeline created in the previously mentioned method [10]. his was one of the reason why we added a short description of its characteristic here, and why we mostly discussed methods that require depth information to work properly. The following chapter describes method [10] in-depth.

Model Globally, Match Locally

After researching the existing methods of pose estimation we have decided to base our further research on the method described in [10]. Figure 2 shows the block diagram of this method.

Model Reference and **Scene** blocks simply represent the input point clouds of the points and their normals. All other bocks represent actual processes and will be described in the the following sections. Point Pair Features are used in all of these blocks. Their description will be given first.

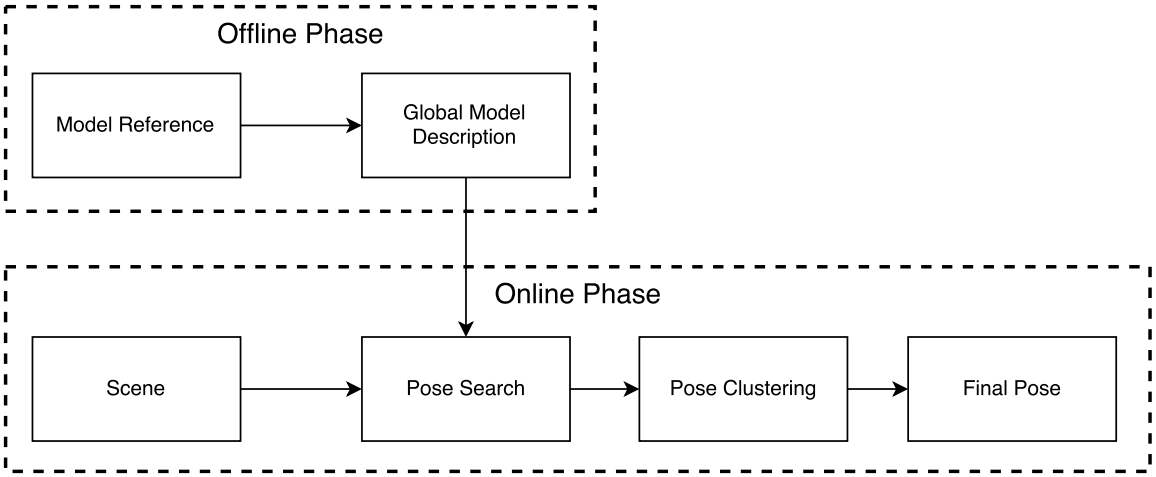


Figure 2: Model globally, match locally block diagram

Point Pair Feature

Point Pair Feature (PPF) used in this method describes relative position and orientation of two oriented points. For two points p_1 and p_2 with corresponding normals n_1 and n_2, we can define the PPF as:

$$ppf(p_1, n_1, p_2, n_2) = (||v||_2, \angle(n_1, v), \angle(n_2, v), \angle(n_1, n_2)), \tag{1}$$

where vector v is defined as $v = p_1 - p_2$ and $\angle(a, b) \in [0, \pi)$ represents angle between two vectors. Figure 3 illustrates this.

Such PPF definition is not suitable for the comparison of the point pairs between model reference and scene because of its low error tolerance. Therefore, $||v||_2$ and $\angle(n_1, v), \angle(n_2, v), \angle(n_1, n_2)$ should be discretized. Our experimentation showed that integer range $[0, 255]$ is usually sufficient for representing all of the values. This allows us to store PPF as a single 32-bit number which will make PPF comparisons very fast and easy to transfer (OpenCL device).

Step size of the distance can have this low range because it represents the number of possible distance steps in the model reference, not in the scene. Therefore, if we are searching for a small model in a big scene we can still have relatively small distance steps and calculate distance

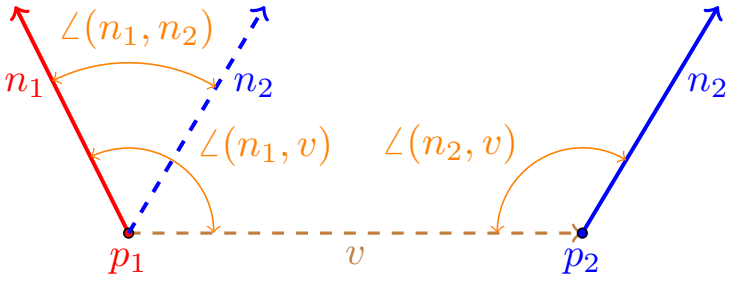


Figure 3: Point Pair Feature

in higher resolution first, then reject all point pairs with distance greater than our range for distance because such points cannot both belong to the model reference.

PPF Space

Finding of the object pose can be reinterpreted as an ability to transform point p_r that belongs to the object with known pose (model reference) onto point p_s that belongs to the identical object, but with an unknown pose (object inside the scene). Figure 4 shows how a point pair is transformed by a transformation $\mathrm{T_{a\to b}}$ from the objects a (model reference) onto the object b (object inside the scene).

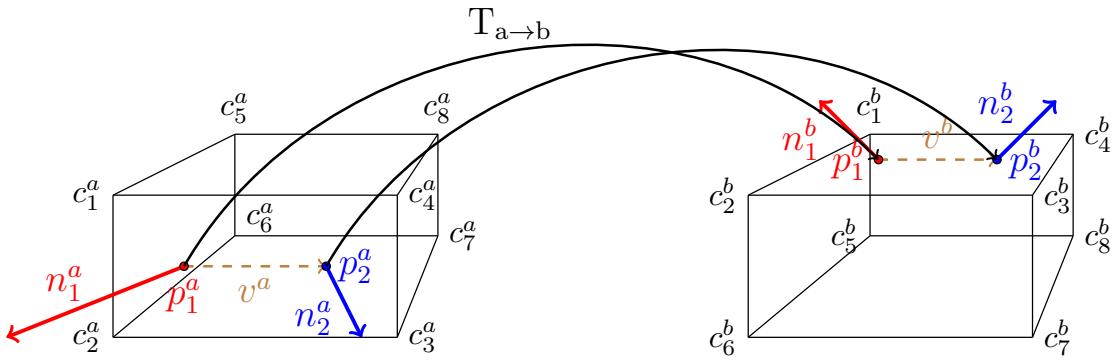


Figure 4: Transformation of a point pair from the object a (model reference) onto object b (object inside the scene).

To obtain this transformation intermediate coordinate system is used. We denoted this coordinate system as PPF space. PPF Space is orthogonal 3D coordinate system where:

origin corresponds to the position of the first point of the point pair.

z axis corresponds to the normal of the first point.

y axis is selected as one of the positive or negative axis of the original coordinate system. Which axis will be selected is determined by calculating dot products of the normal that belongs

to the first point and all negative and positive axes of the original coordinate system and then selecting the axis with the lowest absolute value of the corresponding dot product. This selected axis then has to be adjusted to be perpendicular to the previously obtained z axis.

x axis corresponds to the cross products of the previously obtained z and y axes.

Figure 5 shows identical point pairs on identical objects that differs only in their poses. Figure 6 shows objects and point pairs from the Figure 5 transformed into the PPF space. If we want to transform point p from the object a onto the same point on the object b, we need to know a point pair on each object so that the two point pairs represents the same points. Such point pairs are denoted by points p_1, p_2, normals n_1, n_2 and vectors v in the figures. We then transform point p into the PPF space of the point pair of the object a. And because the transformed point pairs in the PPF space of the objects a and b are identical, we can perform inverse transformation from the PPF space of the object b into the global space of the object b.

This method alone will not work on objects a and c because their point pairs transformed into the PPF space differs. To make them identical one of the point pairs needs to be rotated around its z axis. Therefore, we need to measure the angle $\angle v$ between the v vectors of the objects a and c transformed into the PPF space with their z dimension set to 0. To transform point p from object a to object c, we will apply the same method as described before. However, we will rotate the point p in the PPF space of object a by the angle $\angle v$ before we perform its inverse transformation into the space of the object c. Figure 7 represents this process. Matrix $\mathrm{T_{a\to c}}$ represents the pose, matrices $\mathrm{Ps_a}, \mathrm{Ps_b}$ represent transformation matrices into PPF space for the objects a and c respectively, matrix $\mathrm{R_z(\angle v)}$ represents rotation matrix around axis z by angle $\angle v$ which represents angle difference of the v vectors transformed into the PPF space. Vectors v^a_{z0} and v^b_{z0} represent the vectors v_a and v_c with z dimension set to 0.

The fact that we need to measure the angle difference between the v vectors of the paint pairs in the PPF space makes PPFs with the parallel vectors n_1 and v unusable. Such PPFs should be discarded.

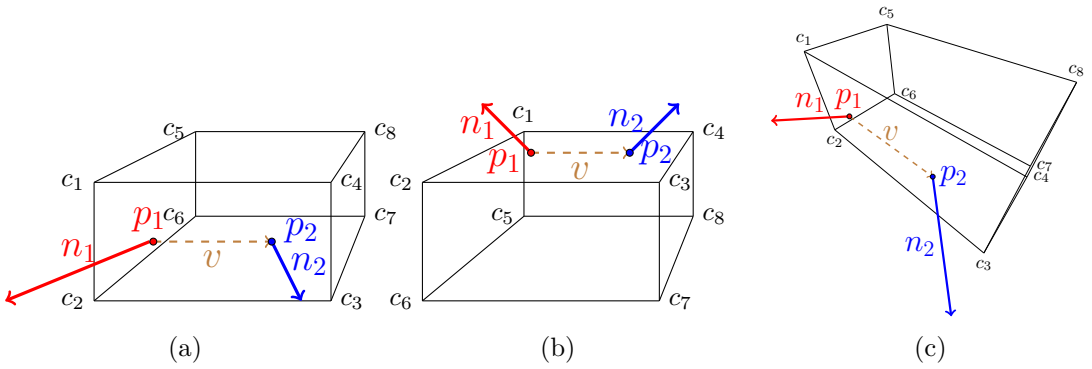


Figure 5: Identical point pairs on identical objects with different poses.

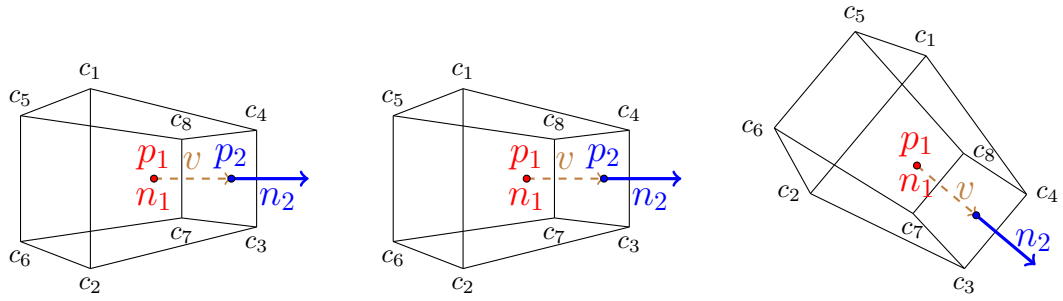


Figure 6: Objects from the Figure 5 transformed into PPF Space.

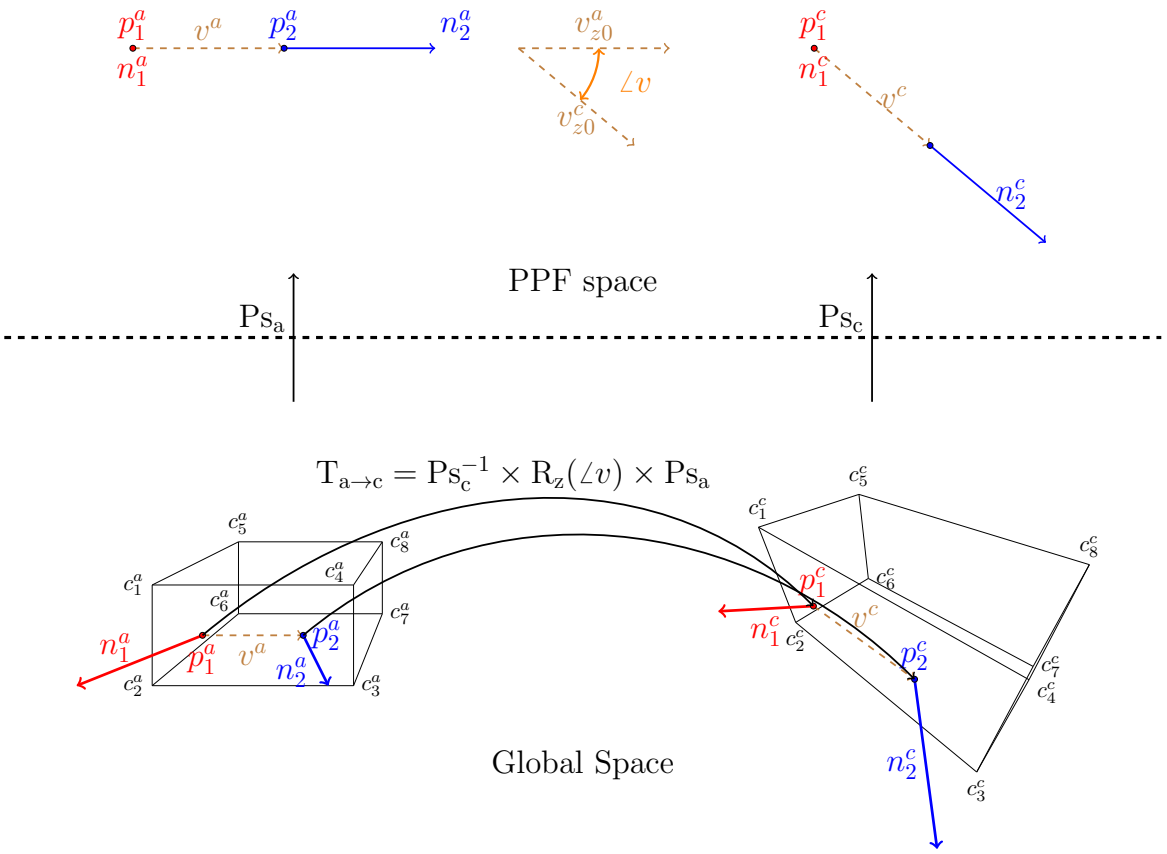


Figure 7: Final transformation of the point from the object a onto the object c.

Offline Phase

The process described in the previous section assumed that we know such point pair on each object, that these two point pairs represent the same points. However, first we will have to discover such point pairs through an identical process but split into two phases for increased efficiency: offline phase that only uses the model reference and online phase that requires both the model reference and the scene data.

To find the identical point pairs we will have to perform comparison of PPFs of the point pairs in the scene with the PPFs of the point pairs in the model reference. We will begin by preparing a list of all unique PPFs in the model reference during the offline phase. Such list should be stored in a container with a fast element search. However, multiple point pairs can have the same PPF. Figure 8 shows a situation where a point p_5 will have the same PPF with the points p_2, p_4, p_6, p_8 and other points as well will have the same PPF with multiple points. This shape represents a fraction of any bigger planar area on the reference model or the scene. Therefore, multiple points with the identical PPF is a normal occurrence.

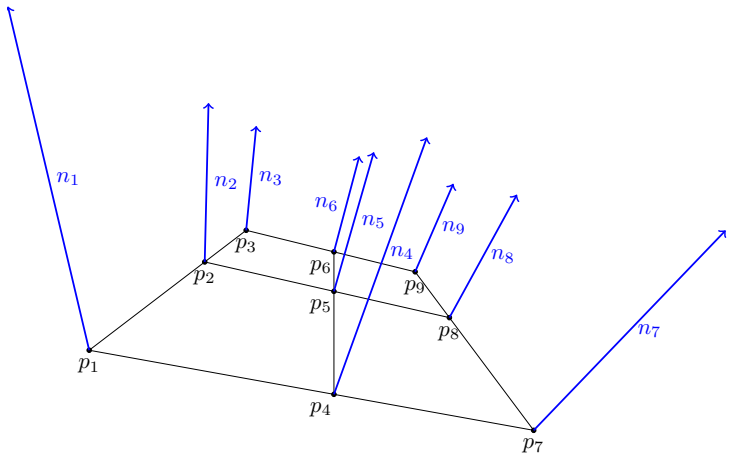


Figure 8: Shape where multiple point pairs will result in the same PPF.

Each unique PPF in the previously mentioned list will have to contain another list with all point pairs that have that PPF. However, the only reason why we needed the second point of the point pair in the method to transform point from the first object (model reference) onto another object (scene) was to obtain v vector of the point pair that belongs to the first object and that vector is needed only to calculate angle difference between that vector transformed into PPF space and v vector of the point pair of the other object transformed into PPF space, both with z dimension set to 0. Instead of obtaining that angle difference from these two v vectors directly, we will calculate the angle difference between each v vector with its z component set to 0 and y axis of the PPF space separately and then subtract these differences to obtain the same result.

With this separation we can precalculate all angle differences for every point pair for every unique PPF in the model reference during the offline phase and store them and the index of the first point of that point pair into a list, to which its unique PPF points to. By doing this, we will obtain a structure depicted in the Figure 9 which we call global model description.

Global model description is then used in the online phase to find the matching points between the model reference and the scene and the poses of that matching points.

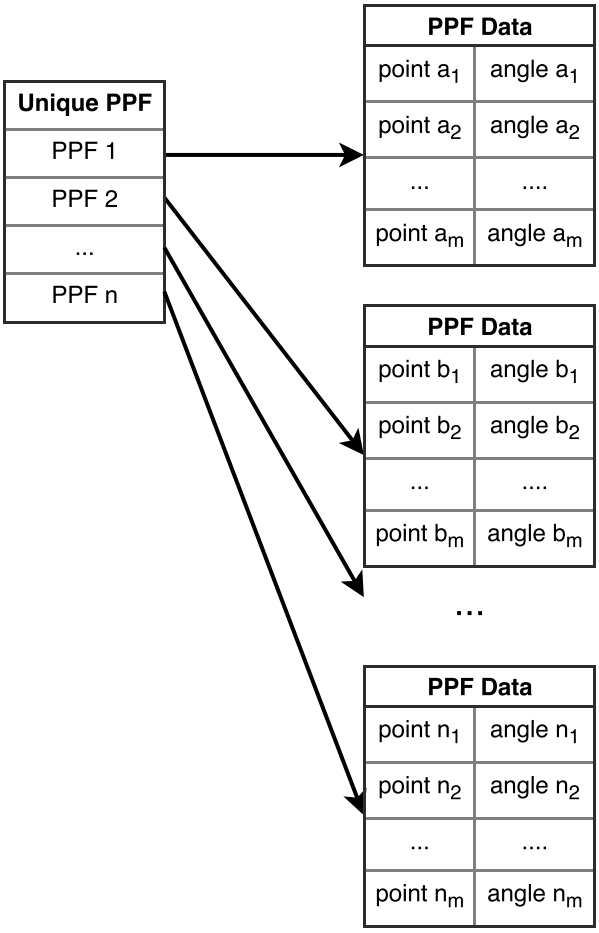


Figure 9: Global model description

Online Phase

To obtain the pose of the object in the scene we first must find points in the scene that we can map onto the model reference. From the previous section it is obvious that simply calculating a PPF is not enough to identify the same point pair because multiple point pairs have the same PPF. Therefore, we will calculate PPFs for multiple point pairs with the same first point and use the global model description to find all the lists of the first point indices and angles for each PPF and cross reference them to choose the point index and angle that appears on the most lists and calculate the pose from them. We refer to this process as pose voting.

Pose Voting

The goal of the pose voting is to determine which point index and angle difference should be used for deriving the pose of the point on the object. To achieve this we will accumulate all possible point indices and angles in a 2D array. Angle is discretized to allow its usage in the array, and to eliminate its low error tolerance.

- Each row in the accumulator array represents one point in the model reference.
- Each column in the accumulator array represents one angle step of the angle difference.

To estimate the pose of the point p_s the following steps must be taken:

1. Zero the accumulator array.
2. For each point pair, where point p_s is the first point do:
   - 2.1 Calculate PPF of the point pair.
   - 2.2 Obtain PPF Data from the global model description for previously calculated PPF. If the PPF is not in the global model description, choose new point pair and start again.
   - 2.3 Measure angle difference a_s between v vector transformed into PPF space with its z dimension set to 0 and y axis of that space.
   - 2.4 For each element in the obtained PPF Data do:
     - 2.4.1 Subtract angle a_s from the angle in this PPF data element to get the angle difference.
     - 2.4.2 Map angle to $[0, \pi)$ interval.
     - 2.4.3 Calculate the index of the angle difference based on the size of the angle step of the accumulator.
     - 2.4.4 Increase value in the accumulator on the row that corresponds to the first point index of this PPF data element and column that corresponds to the index of the angle calculated in the previous step.
3. Find the the element in the accumulator with the highest value and record its row and column index.
4. Find the point p_m in the model reference that corresponds to the row index of the highest accumulator value.
5. Transfer column index of the highest accumulator value back to the angle difference a_d based on the size of the angle step of the accumulator.
6. Based on the points p_m and p_s, their normals and angle difference a_d construct the pose of the point p_s.

Pose Clustering

From the previous section it is obvious that we can obtain pose of the model reference for each point in the scene. Unfortunately, some of the poses obtained in this way will be incorrect. To

pick the right pose we will cluster all the obtained poses into clusters that contain similar poses and then select the one having the most poses as the best representation the actual pose of the model reference in the scene. To obtain the final pose, we can simply pick a random pose from the best cluster, but more recommended method is to average all poses inside that cluster in order to obtain more accurate final pose.

Implementation

Targeted Architectures

Our implementation utilizes both multi-core and many-core architectures during the online phase. Offline phase uses only single-core architecture, this decision was made, because the speed of the offline phase was not our primary concern.

- C++11 threads are used for multi-core architectures, they were selected because our implementation is multi-platform.
- OpenCL is used for many-core architectures and was successfully tested on:
  - NVIDIA GeForce 820M
  - NVIDIA GeForce 920M
  - NVIDIA GeForce GTX 960M
  - NVIDIA GeForce GTX 1070
  - Intel® Core™ i5-5200U

Rasterization of the Depth Maps

Our implementation uses depth maps as scene inputs, depth maps are already rasterized. However, they are from devices, such as depth cameras, that use perspective projection. Therefore, with such perspective projections actual vertical and horizontal position of the point is dependent on its depth. That is very inconvenient, because we have no fast way of estimating which point corresponds to which vertical and horizontal position in the scene.

To solve this issue we first convert such depth maps back into 3D space, and then rasterize them into a new depth map, where vertical and horizontal distance between the pixels are constant. In our implementation vertical and horizontal distances between the pixels are always identical.

One of the benefits of the rasterization is the fact, that we have precise knowledge of the minimal distance between the points that is useful for the pose estimation (PPF discretization), and therefore we can lower the resolution of the depth map.

Another of the benefits is that when we want to find the pose of a point in the scene, we can very easily find the area of the points around that point with vertical/horizontal 2D distance smaller than the maximal distance between the point in the global model description. We know this would be possible even with the original depth map, but the solution would be more complex and sizes of the areas would differ, which would make planning of the execution on many-cores architectures more complex.

Another benefit is that the calculation of the vertical and horizontal position of the point is done by simple multiplication of its x and y index by the grid size, and for example x and

y distance between two points can be calculated by subtracting their x and y values first, and then multiplying them by the grid size, which would not be possible in the original height map. This makes using a four channel float texture to represent normals and depths of the points in the OpenCL kernels more appealing.

Filtering of the Depth Maps

As mentioned in the previous sections depth maps are obtained from devices such as depth cameras. Such depth maps usually contain noise and areas where device failed to measure the depth. Figure 10 shows depth map after rasterization, which contains both of the mentioned phenomenas. The reason why we first rasterize depth maps even with such defects is because rasterization usually lowers the resolution of the depth map, and therefore acts as a filter itself, and with lower resolution of the depth map all additional operations on it will be faster and more consistent because the distances between the points are identical.

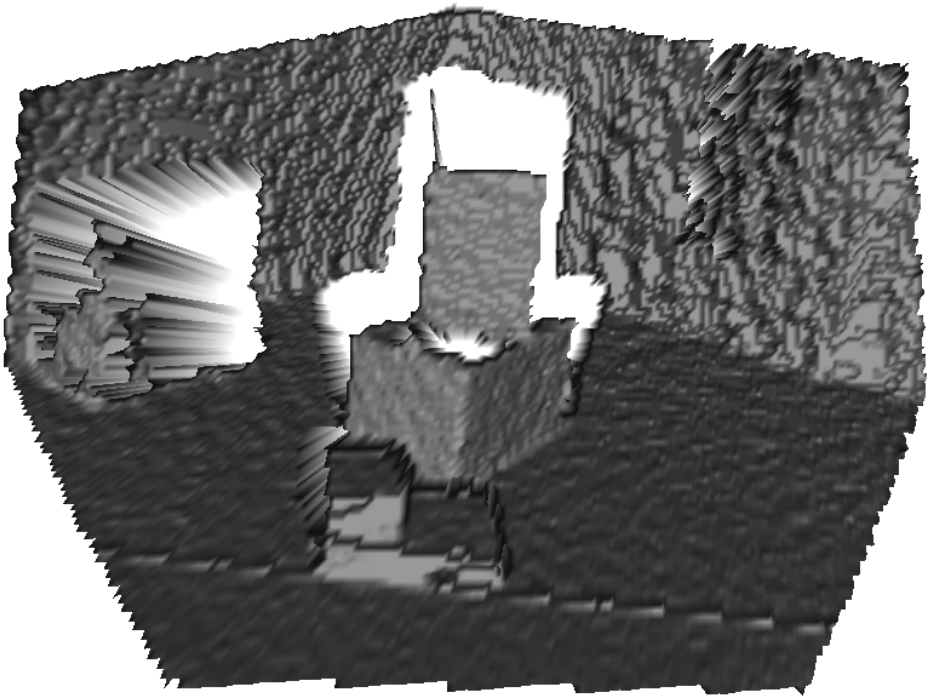
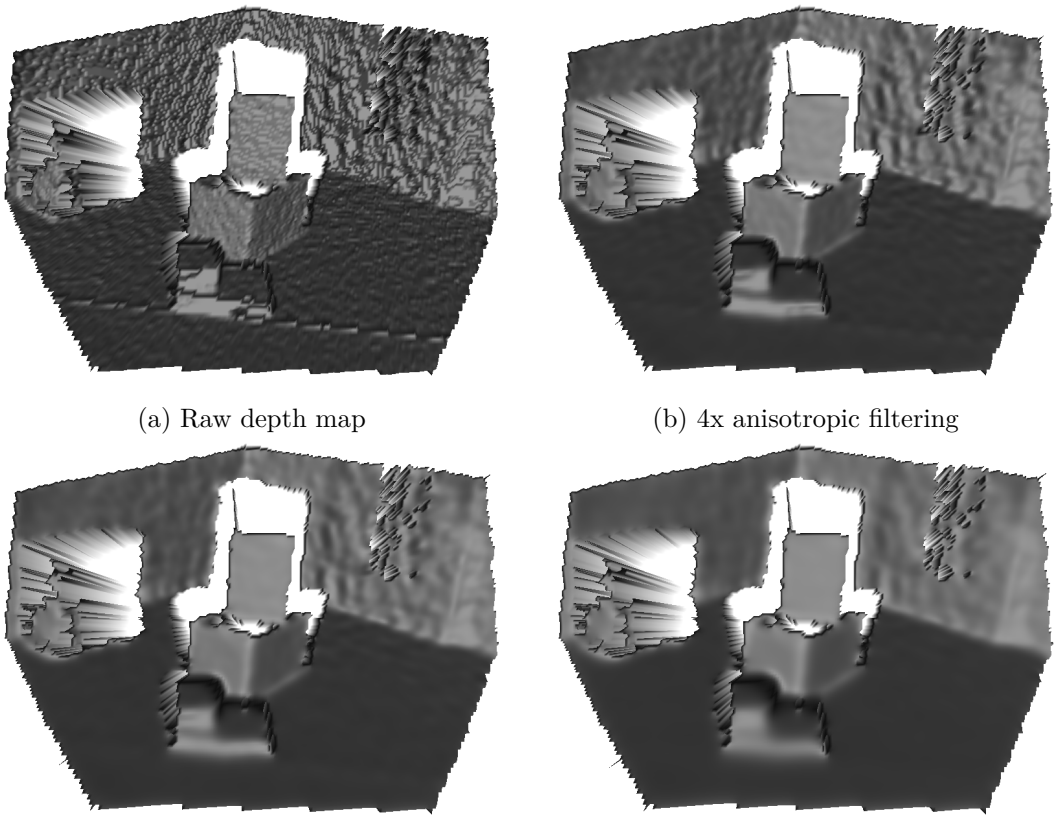


Figure 10: Depth map after rasterization with no additional processing.

As with most of the other part of our implementation depth map filtering have to be fast and scale well on many-cores architectures. Therefore, we have decided to use anisotropic diffusion. Figures 11b 11c 11d show the depth map from the Figure 10 after 4, 8 and 16 pases of anisotropic filtering respectively.

It is obvious that filtering will further distort the edges of the object. However, pose estimation method that we use does use object edges for the pose estimation. Therefore, this downside



(a) Raw depth map

(b) 4x anisotropic filtering

(c) 8x anisotropic filtering

(d) 16x anisotropic filtering

Figure 11: Anisotropic filtering of the depth map

of the filtering is less relevant. And due to the heavy noise in the depth map, the edges were distorted even in the original data.

Hole Filling of the Depth Maps

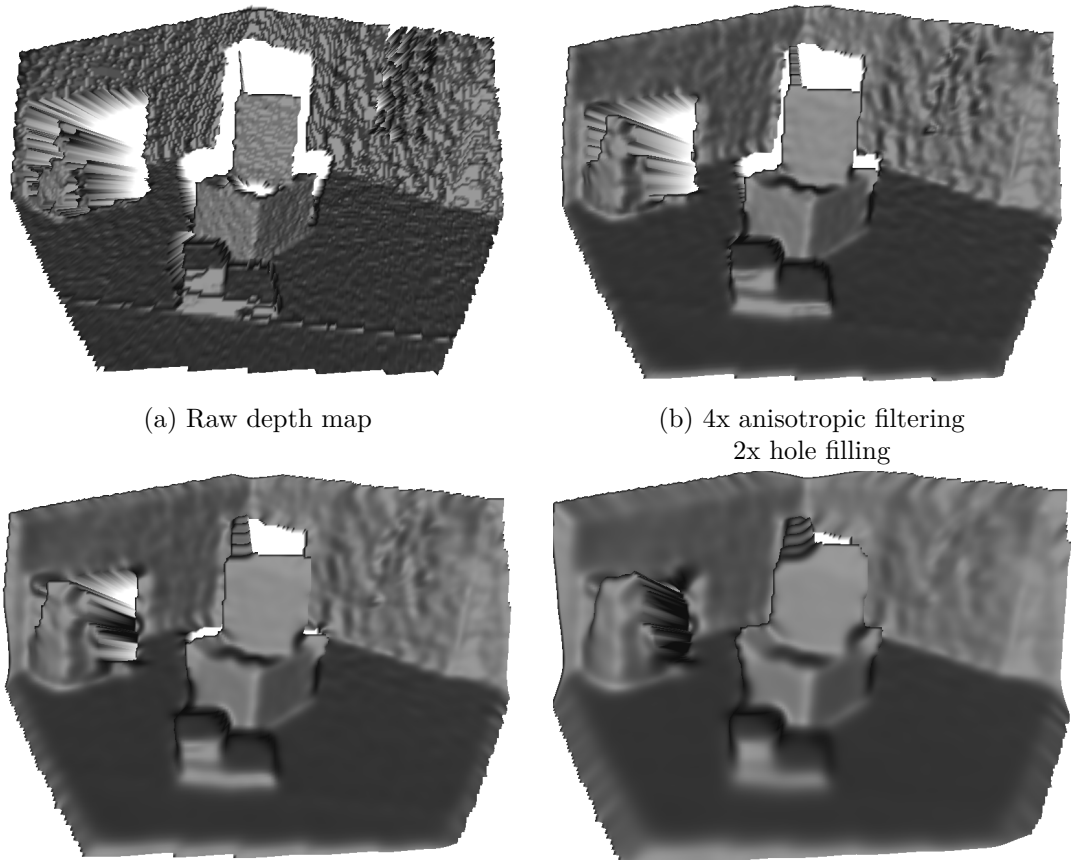
First important decision during the approximation of the invalid depth values, is if we want approximate all the missing values or not. We have come to a conclusion, that trying to approximate depths in the large areas where the data are missing is contra productive, because such approximation has very little information to be based on, and therefore its accuracy is low and the usage of such data will most likely have no contribution to the pose estimation. One of the examples of such situation is the almost fully uncaptured object on the left side of the Figure 10. On the other hand, the invalid depths on the wall to the right should be filled in, because there is enough valid depth information surrounding them.

Based on this decision, we have decided to use very simple method of hole fitting. We simply modified our anisotropic filtering kernels to perform anisotropic filtering when the filtered point has valid depth, and if it does not, then will simply set its depth to the average of the valid depth in the 8 surround depth values. If no valid depth is found within these values, then the

depth stays invalid. This method is very simple, and therefore has very low impact on the overall perforce. We include this feature only into the beginning N iterations of anisotropic filtering, and keep the remaining ones without it to assure anisotropic filtering of the newly added depth values.

Following figures represent the depth map from the Figure 10 after various numbers of anisotropic filtering pases, and various numbers of first anisotropic pases, that also had previously mentioned hole filling addition in them:

- Figure 12b 4 total pases, 2 hole filling pases.
- Figure 12c 8 total pases, 4 hole filling pases.
- Figure 12d 16 total pases, 8 hole filling pases.



(a) Raw depth map

(b) 4x anisotropic filtering
2x hole filling

(c) 8x anisotropic filtering
4x hole filling

(d) 16x anisotropic filtering
8x hole filling

Figure 12: Anisotropic filtering and hole filling of the depth map

From the section above and the boxes in the Figure 12d it is obvious that this simple hole filling method is not suitable for filling of large areas. On the other hand, even 2 pases of hole

filling were able to fill the holes on the wall in the Figure 12b. Therefore, for this particular scene 2 or 3 iterations of hole filling would be considered optimal.

This simple hole filling method is also very deterministic. Each iteration can calculate the depth approximation, if there is at least one valid depth in the 8 surround depth values and the distance between the depths is fixed, and therefore in a typical case when the hole is surround by the valid depths one iteration is guaranteed to fill the holes with the maximal radius of the raster size. Therefore, N iterations of hole filling is guaranteed to fill fully surrounded holes with the maximal radius of N raster sizes.

Restricting Point Pair Features

PPFs that were defined in the Section 3.1 can be used in any 3D scene. However, our application will only work with scenes obtained from depth maps, where points pairs with 180 degrees angle between their normals are due to the original perspective projection of the depth map impossible, unless the normals are incorrect, furthermore we can also omit point pairs with high angle between their normals (for example 150 degrees), because to obtain such angle at least one of the points has to have a high angle between its normal and inversed camera direction, and such normals are very imprecise due to the noise in the depth map. Therefore, we consider PPFs with high angle between point normals $\angle(n_1, n_2)$ to be invalid.

We have also come to a conclusion, that PPFs with low angle between point normals $\angle(n_1, n_2)$ are very abundant in the scenes, because they represent a plane, this was already mentioned in the Section 3.3, where Figure 8 illustrates this situation. Such PPFs describe a large number of different point pairs, which will result in a large number of votes for different points in the accumulator, and therefore mainly increase the number of votes each reference model point and angle step in the accumulator receives. Such behavior does not have a significant contribution to finding the best candidate in the accumulator and causes memory access bottlenecks, especially on many-cores architectures. Due to this fact, we can reject such PPFs in the offline phase, and by doing so reduce the size of the global model description.

Color Point Pair Feature

PPF that also contains color information for both points was, for example, described in [7]. Color information inside the PPF will reduce the number of the point pairs with the same PPF. Such PPF is define as:

$$ppf(p_1, n_1, p_2, n_2) = (||v||_2, \angle(n_1, v), \angle(n_2, v), \angle(n_1, n_2), c_1, c_2)\,, \tag{2}$$

where colors c_1 and c_2 represent the colors of the first and second point of the point pair respectively, and all other variables are identical with the ones in the original PPF definition (Section 3.1).

Encoding of the color must be robust enough to handle illumination changes of the scene. Therefore, RGB color space is not suitable. HSV color space was used in [7]. However, standard HSV color space does not work well with colors that have low saturation value (shades of gray). For such colors even small changes in one of the RGB channels may lead to big change in Hue. Therefore, Hue should be used only if the saturation is high enough. All of the components have to be discretized.

Because of the insufficient testing, we were not able to determine optimal setting for the discretization, but we have discovered, that 16bit is more than sufficient space for the color representation.

Therefore, in out implementation we use 64bit PPF, where the first 32bits is used for the PPF representation, that was described in the Section 3.1, and the two remaining 16 bit groups are used for colors of the point pair.

Global Model Description

Our primary concern when implementing the global model description was its portability between the C++ code and OpenCL kernels. Therefore, we initially decided to use two simple arrays to represent it:

1. Unique PPFs Array - sorted array that holds the unique PPFs. Each item in this array contains:

   PPF which is used for sorting of this array.

   PPF data start index represents index of the first item in the **PPF Data Array**, that belongs to this PPF.

   PPF data end index represents index of the first item in the **PPF Data Array**, that **does not** belong to this PPF.

2. PPF Data Array - array that holds list of all indices of the first points and angles for all point pairs with valid PPF.

This solution worked. However, we were concerned with the impact of the binary search on GPU performance, because each step of this search requires access to different part of the global memory, and therefore is not cache friendly. This lead us to use of read only hash table with 64bit hashes, which is represented by 3 simple arrays:

1. Hash Bucket Index Array - array where index in the array corresponds to the remainder after integer division of the PPF hash value and size of this array. This array contains:

   PPF hash bucket start index represents index of the first item in the **PPF Hash Bucket Array**, that contains PPF hashes which will have the same integer division reminder.

PPF hash bucket end index represents index of the first item in the **PPF Hash Bucket Array**, that **will not** have the same integer division reminder.

2. PPF Hash Bucket Array - array that holds the unique PPFs. Each item in this array contains:

   PPF of this item.

   PPF data start index represents index of the first item in the **PPF Data Array**, that belongs to this PPF.

   PPF data end index represents index of the first item in the **PPF Data Array**, that **does not** belong to this PPF.

3. PPF Data Array - array that holds list of all indices of the first points and angles for all point pairs with valid PPF.

With this structure, we only have to make at most 3 random memory accesses to obtain the PPF Data for the PPF:

First into the **Hash Bucket Index Array**. if the PPF is not in the global model description, then this will be our only random memory access. If we were to use binary search, we would have to do $\lfloor \log_2(N) + 1 \rfloor$ random memory accesses when the PPF is not in the global model description.

Second into **PPF Hash Bucket Array**, where we will do comparisons of the PPF hashes. All comparisons will require only additional sequential reading of the memory.

Third into **PPF Data Array** itself.

We also store the maximal distance between the points in the model reference, which is later used to reject calculation of the PPFs between the points in the scene, that have greater distance than this maximal distance. This saves the need to search for the PPFs, that cannot exist in the global model description.

Selection of the Points For the Pose Estimation

As already mentioned in the Section 3.3, pose can be estimated from each point in the depth map and theoretically one well selected point on the object which we want to estimate the pose will give us the correct object pose. However, we do not know where the object is in the scene, and also some points on the object will give us invalid pose estimations, and therefore clustering is used to select the correct pose. This implies that we do not need to estimate the pose of every point in the depth map to obtain the pose. It is obvious that the more points are used during the pose estimation the more robust and time consuming the pose estimation is.

The goal of our implementation is to work with both static scenes and video sequences. Therefore, we use two different methods of selecting the points for the pose estimation:

Constant method simply selects every nth point in the rasterized depth map in both axis. Red dots in the Figure 13 represent the points selected for the pose estimation by this simple method. The number of n skipped points is given by user, because it is scene and model dependent. This simple method is used for the static scenes and frames of video sequences, where we do not know the pose from the previous frame.

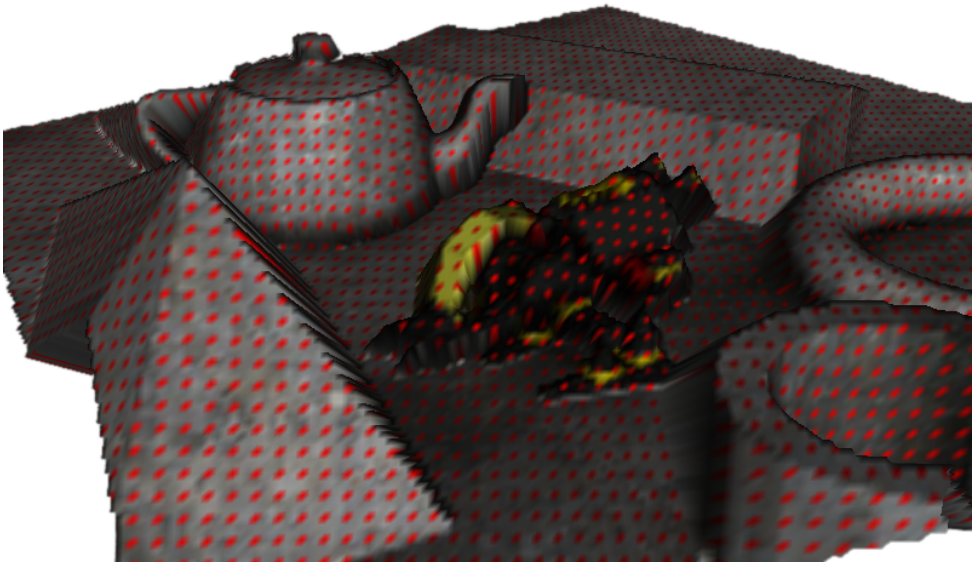


Figure 13: Selection of points for pose estimation, constant method.

Adaptive method is used for the frames of the video sequences, where we know the pose of the object in the previous frame. This method assumes that the estimated pose in the current frame will be similar to the pose in the previous frame. In this method n points for the pose estimation are obtained by randomly selecting values with normal distributions for their x and y coordinates. Parameters of the normal distributions are obtained by following these steps:

1. Calculate axis-aligned bounding box(AABB) of the model reference.
2. Calculate center of the AABB.
3. Use pose transform from the previous frame to transform the corners and the center of the AABB to the scene.
4. Set median μ of the normal distributions for the x and y coordinates to the x and y values of the center of the AABB in the scene.
5. Calculate new AABB from the corners of the old AABB in the scene.

6. Set standard deviation σ of the normal distributions for the x and y coordinates to $\sigma = coef \cdot \frac{size}{2}$, where $size$ is the size of the AABB in the axis, that corresponds to the coordinate of the normal distribution and $coef$ is a constant. Our default values is $coef = 0.3$.

The number of selected points n is given by user, because it is scene and model dependent. Red dots in the Figure 14 represent the points selected for the pose estimation by this method. In this case 250 points were selected.

Figure 14: Selection of points for pose estimation, adaptive method.

Reference Model Culling

Adaptive selection of the points for the pose estimation from the previous section works with the assumption, that the pose of the object in the current frame of the video sequence is similar to the pose in the previous phase. This assumption can also be used to reduce the size of the global model description by removing occluded points of the reference model. To determine if the point in the model reference is occluded, angle between its normal and camera view direction is calculated and if that angle is higher than 90 degrees, then the point is considered to be occluded.

The goal of reference model culling is to reduce the size of the global model description, and therefore it has to be used in the offline phase. This implies, that a set of culled model

references has to created for such view directions, that it is possible to find a culled model, that is sufficiently close to the culled model obtained with arbitrary view direction.

In our implementation we use 18 view directions for culling, y axis is up:

- top
- 4 90 degrees rotations around y axis with 45 degrees rotation around x axis
- 8 45 degrees rotations aounnd y axis.
- 4 90 degrees rotations aounnd y axis with 45 degrees rotation around x axis
- down

Figure 15 illustrates this views.

The global model description that will be used for the pose estimation is then selected from these 18 culled global model description based on the minimal angle between the view direction of the culled global model description and the view direction obtained from the previous pose. Culled reference model is by its definition a subset of the original reference model. Therefore, the the accumulator of the whole global model description can be used for the culled ones by simply not processing the unused part of it.

Trigonometric Functions Approximations

After some profiling of our CPU implementation of the pose calculator we have discovered, that in certain situation PPF creation took more time, than PPF search and filling of the accumulator.

PPF creation from the point pair is relatively fast operation the most time consuming part is the calculation of the angles $\angle(n_1, v), \angle(n_2, v), \angle(n_1, n_2)$. These angles are only used for comparison of the PPF and no arithmetic operations are performed with them.

Normals n_1, n_2 are always stored in normalized forms and the length of the vector v is part of the PPF. Therefore, its normalization does not add much of a overhead. Therefore, the cosines of the angles are obtained by dot products. Unfortunately, cosines can no be used instead of angles, because of their non-linear nature, which makes them unsuitable for the discretization. Therefore, inverse cosine need to be used to obtain the angle, which is then discretized. Angles are discretized to only 32 or 64 values, which gives us the precision of 5.625 and 2.8125 degrees respectively, and therefore we can use faster approximation of the inverse cosine.

We have decided to use Nvidia's approximation of the **acos** from [1], which is based on the [2]. Listing 1 shows the code in C++. Its maximal absolute error is stated to be smaller than $6.7e-5$ radians (approximately 0.0038 degrees), this makes it more than accurate for our purposes.

In our experiments we did not see any degradation of the pose estimation accuracy. Unfortunately, time improvement of the CPU implementation was in the range of 5% of the total

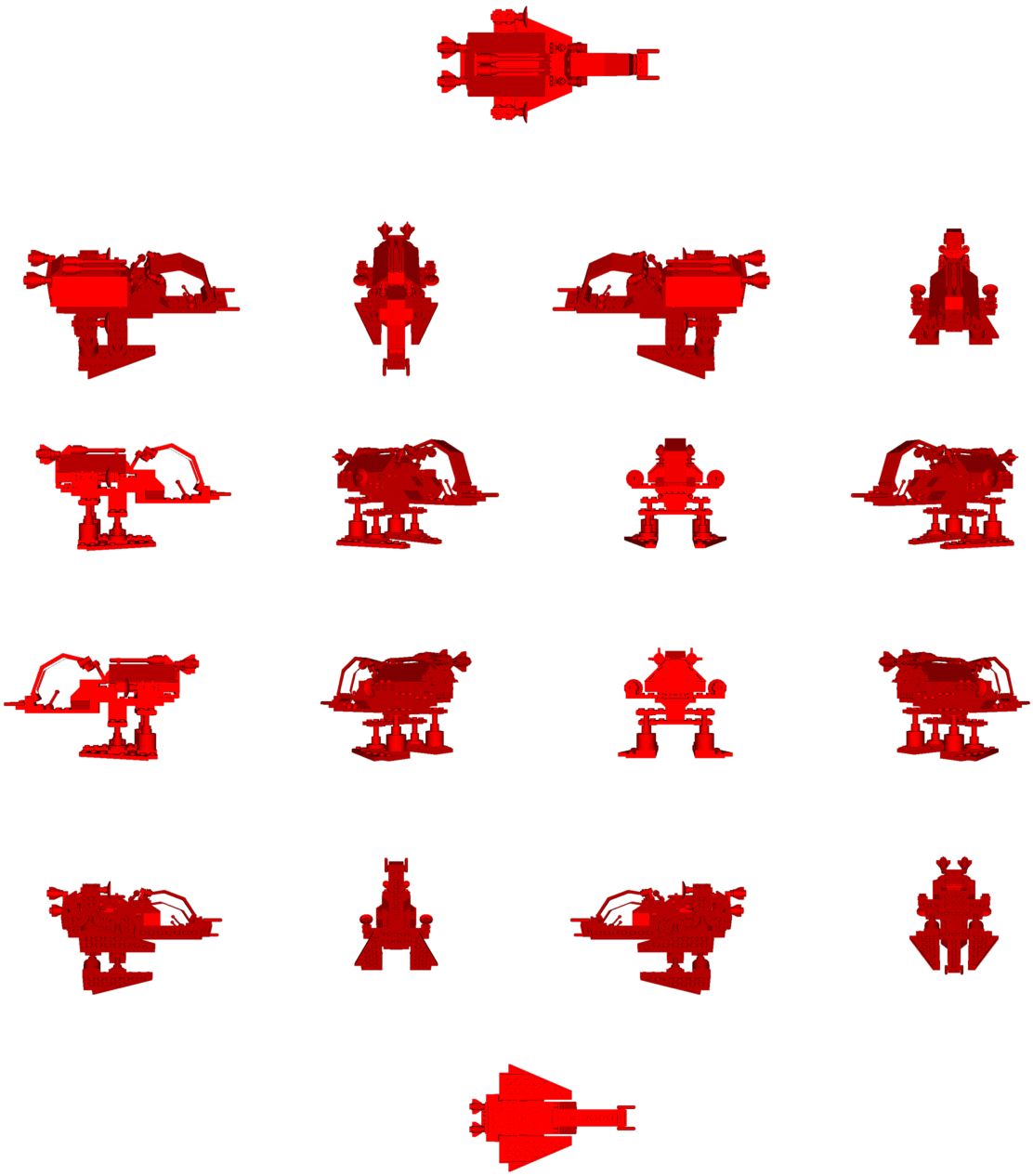


Figure 15: Views used for reference model culling.

time and our OpenCL implementation had no time improvement. Due to this results we have decided to not spend any more time on this topic and also omit it in the rest of this thesis.

```
// http://developer.download.nvidia.com/cg/acos.html
// Handbook of Mathematical Functions
// M. Abramowitz and I.A. Stegun, Ed.
static float FastAcos(float x)
{
    float negate = float(x < 0.0f);
    x = std::abs(x);
    float ret = -0.0187293f;
    ret = ret * x;
    ret = ret + 0.0742610f;
    ret = ret * x;
    ret = ret - 0.2121144f;
    ret = ret * x;
    ret = ret + 1.5707288f;
    ret = ret * std::sqrt(1.0f - x);
    ret = ret - 2.0f * negate * ret;
    return (negate * 3.14159265358979f + ret);
}
```

Listing 1: Acos Approximation

Parallelization

As already mentioned in the Section 4.1 our implementation uses both C++11 threads and OpenCL.

- OpenCL is used almost exclusively during the rasterization of the depth map and its filtering.
- Both C++11 threads and OpenCL devices are used for pose estimation of the points in the scene. This part usually takes most of the time.

C++ and OpenCL pose calculators were implemented and have the same interface. Both of them run on a C++11 thread.

- C++ pose calculator uses its C++11 thread for the work scheduling and pose estimation itself.
- OpenCL calculator uses its C++11 thread for the work scheduling only, the calculation itself is carried out on the OpenCL device, and the C++11 thread itself is used only for the final processing of the results.

We use first come, first served approach for the scheduling of pose estimation work. First all the poses that will be estimated and all the parameters for them are inserted into the work queue. Creation of the work queue is not parallelized, because it is very fast. Each item in the work queue contains:

- Position of the point on which the pose estimation will be carried out.
- Normal vector and depth of the point on which the pose estimation will be carried out.
- Position of the top left and bottom right corner of the rectangular area on which the point pairs will be selected.
- Rotation matrix for transformation into PPF space.

After work queue is created each thread with C++ or OpenCL pose calculator takes certain number of items from the queue to work on. This process is repeated until the queue is empty.

- Threads with C++ pose calculators take a small number (1-10) of work items from the work queue and processes them in sequence during one work cycle.
- Thread with OpenCL pose calculator takes 512 work items in one work cycle and processes them all in parallel during one work cycle.

This very unbalanced number of work items between the C++ and OpenCL pose calculators exist to ensure, that OpenCL device works on many pose calculations in parallel in order to be fully unitized while C++ calculators work on a low number of work items per cycle to avoid long calculations, because such calculations would stall the other threads at the end of the pose estimations.

OpenCL Parallelization in Detail

Global approach of dividing the work on the pose estimation for the point in the scene was described in the previous section. OpenCL device processes 512 pose estimations in parallel, this section will describe how that parallelization is carried out.

To perform 512 pose estimations at the same time we will require 512 accumulators. Accumulator has fixed number of angle splits. Therefore, it is representable as 1D array, which means that it is trivial to represent multiple accumulators as single array.

From the description of the work queue item (Section 4.11), it is obvious that all point pairs will be selected from a rectangle. Dimensions of such rectangles will be identical, with the exception of the rectangles that will be smaller, because they are limited by the borders of the depth map. In our current implementation we use 8×8 work group for the processing of one pose estimation. Each thread inside this 8×8 work group will have to take steps by 8 in x and y position and processes point pairs on the resulting positions. Figure 16 illustrates smaller 4×4 work group processing 7×6 rectangle. It is obvious that using 4×4 work group on 7×6 rectangle is not optimal, because the number of calculations each thread will perform is uneven, but the rectangle size in the real pose estimations will be larger, because we need high number of point pairs to correctly estimate the pose of one point in the scene. For example, in our test

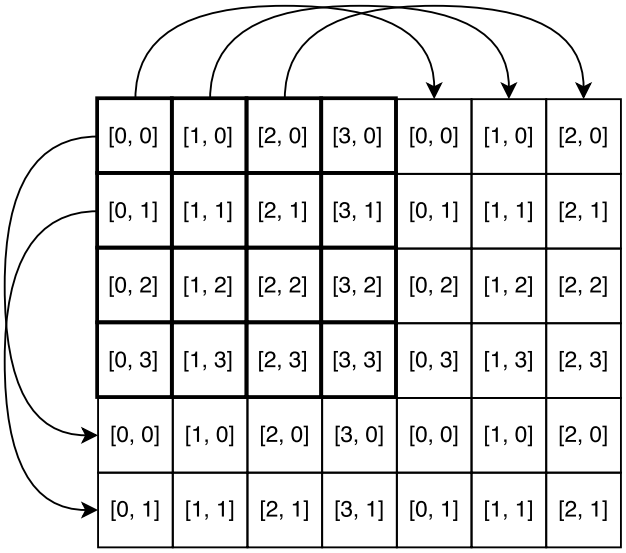


Figure 16: 4×4 work group mapped to 7×6 rectangle

scenes the rectangle size for single pose estimation, that is not clipped by the depth map bounds is 67×67.

It is obvious that the first point of the point pair is in the middle of the rectangle, and therefore the size of any side of the rectangle cannot be an even number. Also one thread will always try to calculate PPF with the first point as both of the points of the point pair and it will be immediately rejected based on zero distance between the points.

This concludes the first stage of the pose estimations, where 512 8×8 work groups fills 512 accumulators. Next stage is almost typical example of performing reduction on many-cores architecture. We identify the highest value and index array index of that value from the values in each row of each accumulator. Figure 17 illustrates typical reduction execution scheme for one work group on many-cores architectures. In this reduction many work groups work on reduction of a large array. Therefore, size of work group can be easily adjusted to the most efficient size for the currently used many-cores device.

In our case we have to reduce a large number of small arrays (64 or 32 elements each), which implies, that one work group will work on only one array. Unfortunately, even in such case size of the work group will still be insufficient for optimal performance on most many-cores devices. To improve the work group size, we have decided to sort multiple arrays by one work group. This is possible because all of the small arrays are stored consecutively in one big array. For this style of reduction we have developed two different execution schemes with different memory access patterns:

Sub-reduction scheme is a straight forward scheme where we simply merge N classical reduction schemes. Figure 18 illustrates this scheme.

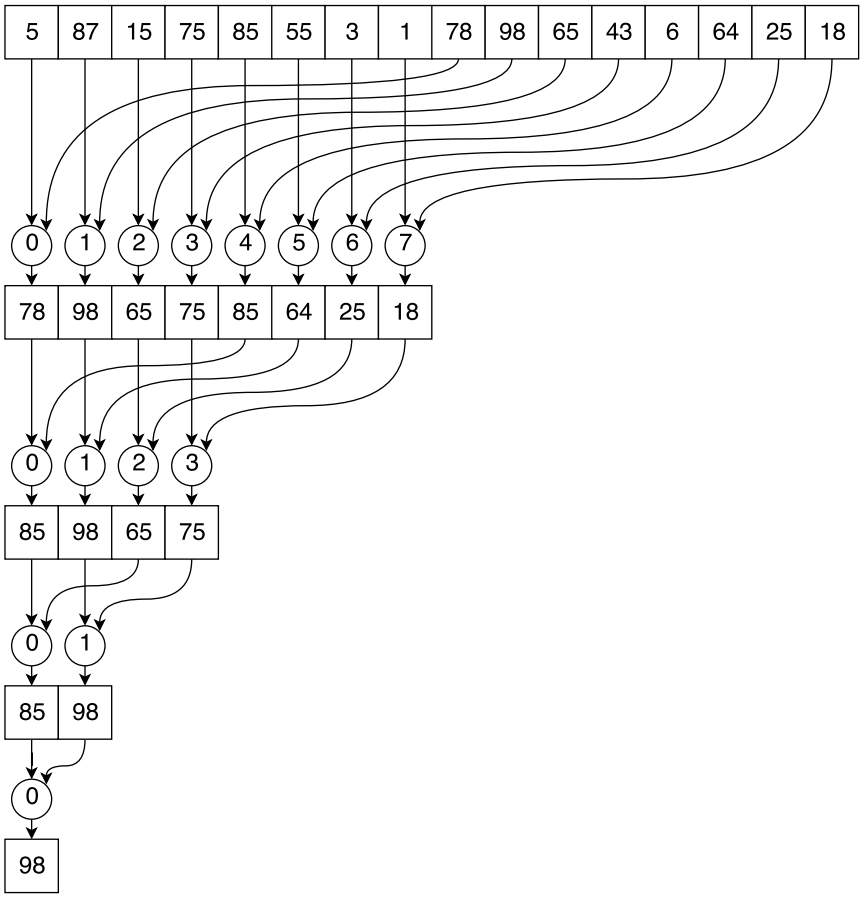


Figure 17: Typical execution scheme of reduction on many-cores architecture.

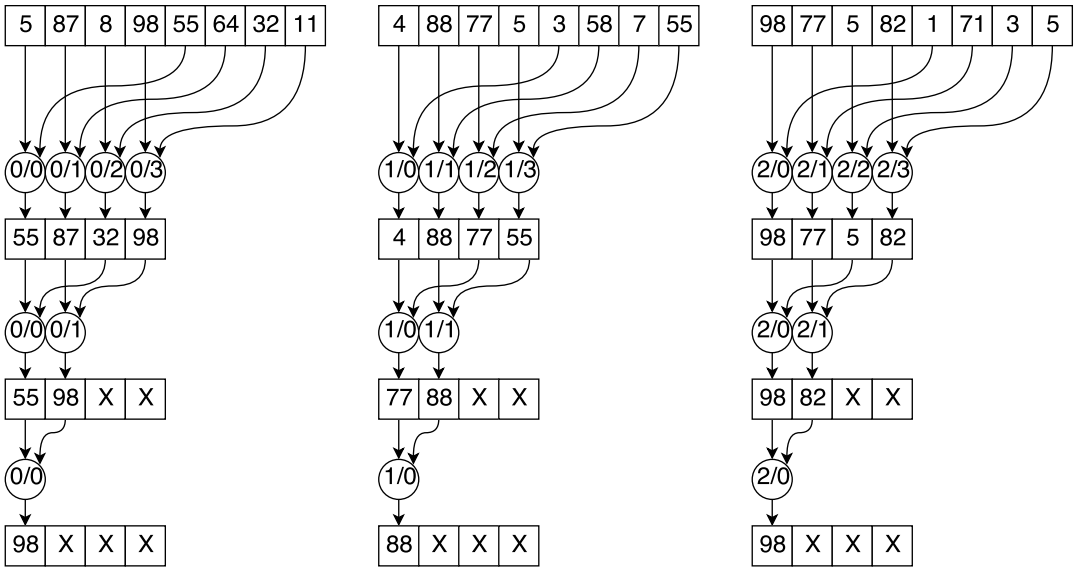


Figure 18: Sub-reduction execution scheme

Swizzle/Reduce scheme is a scheme, that stores the results of the first reduction in such a way, that traditional reduction scheme can be used for the consecutive reductions. Figure 19 illustrates this scheme.

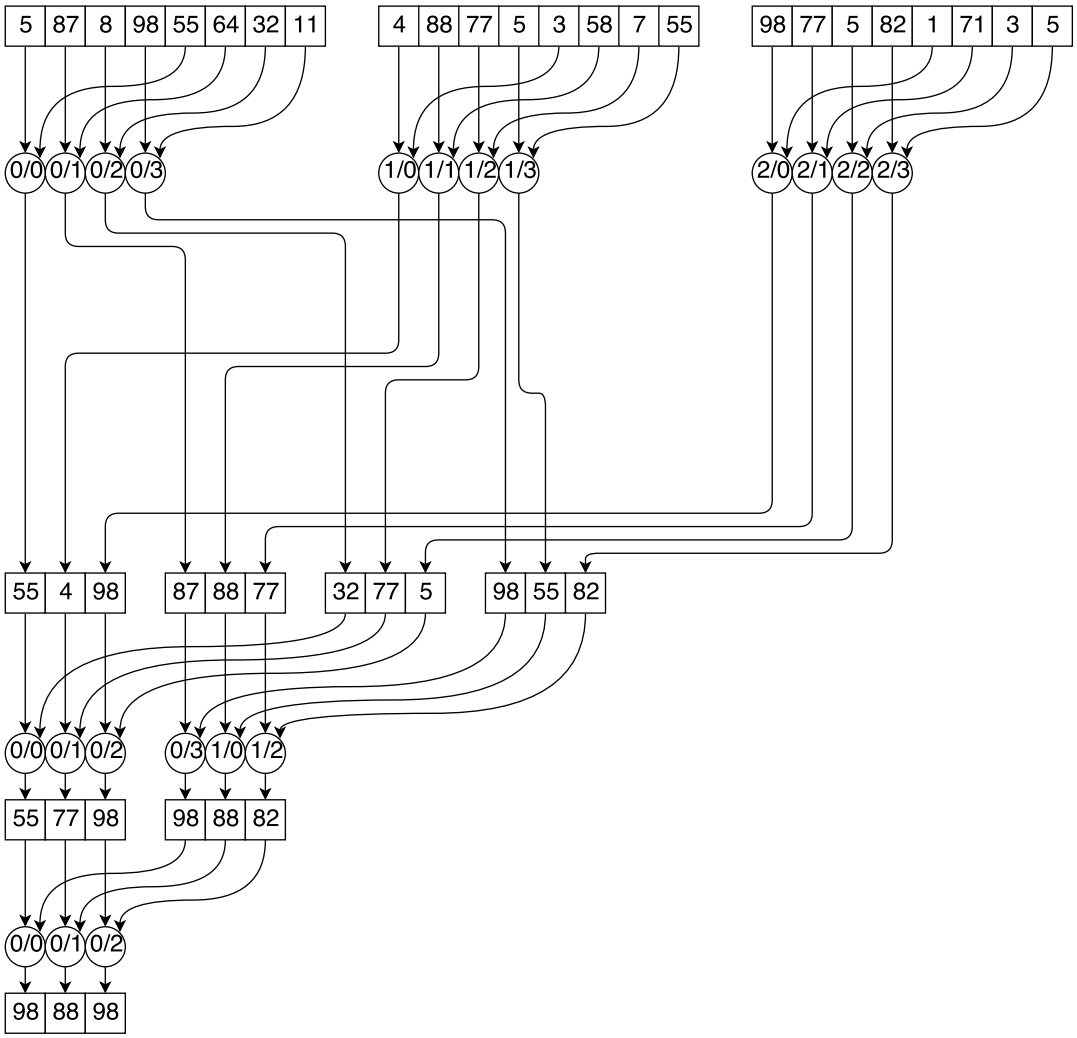


Figure 19: Swizzle/Reduce execution scheme

We have run several test with 64 and 32 elements in the arrays, and sub reduction execution scheme was marginally better (around 2% time improvement) with reasonable number of arrays per work group. Its time degradation with unreasonably high number of arrays per work group was lower than that of swizzle/reduce execution scheme, this is obviously irrelevant, because optimal time can be achieved only with optimal number of arrays per work group. Due to this results we have decided to use sub-reduction execution scheme.

The last stage is the final reduction of the intermediate results form the previous stage during which we will find the highest value from the intermediate results for each accumulator and index of that result, and store it with the index of the highest value form the previous

reduction. These two indices represent the reference model point index and angle index of the pose.

In this reduction the arrays are usually too large to use only one work group for the reduction, but the number of work groups each of the reductions will require is relatively low. For example, if model reference has 1500 points and work group size is 128, and therefore one work group can perform reduction from 256 to 1 value, we will need 6 work groups to reduce 1500 values to 6. Due to this we will need additional step to reduce this low number of intermediate results to one final value.

Due to the fact that the size of the arrays with intermediate results is small and we only have one array per pose estimation of maximum of 512 pose estimations, we do not use reduction at all and simply use each thread to calculate the maximum of each small array. We decided to to not experiment with implementations of this step, because the time it takes is negligible when compared to the total time of the previously mentioned reductions. It is also worth mentioning, that with small enough model and large enough work group one work group is sufficient for the entire reduction. For example work group size is 256 and model reference has 500 points.

Calculations of the final poses from the obtained reference model point index and angle index are all done on the host (Pose Voting steps, Section 5 and 6).

Experiments

This section contains experiments, that were carried our to evaluate various previously described operations. First batch of experiments will be aimed at optimizations, that were discussed in the previous section. Next experiment will evaluate the scalability of our parallelization. Experiment after it will evaluate our final implementation on real scene, that was captured by depth camera. However, before we can start describing the experiments we carried out, we have to describe the frame sequences, that were used. And also explain our methodology for time measurement, and evaluation of the found poses.

Synthetic Frame Sequences

We will use two synthetic frame sequences with two different reference models for the most of our experiments.

Frame Sequence 1

Sought model in our first synthetic frame sequence is LEGO 6876 Alienator. The scene captured in this frame sequence contains only the previously mentioned model. The frame sequence itself contains 360 frames in which the sought model is rotated around its up axis by one degree in each frame. Figure 20 shows 9 evenly selected frames from the sequence. Due to the simplistic nature of the scene it is possible to perform experiments with this frame sequence with a single CPU thread in a reasonable time.

Frame Sequence 2

Sought model in our second frame sequence is a real world mechanical part. The scene captured in this frame sequence contains flat plane on which the sought model is places along with other unrelated models. The frame sequence itself contains 360 frames in which the scene is rotated around its up axis by one degree in each frame. This frame sequence will be mostly used for experiments with OpenCL devices, because its increased complexity makes calculation times with single CPU thread unreasonable.

Measured Times and Ground Truths

The purpose of our first batch of experiments was to evaluate various optimizations, and therefore we measured times of the different parts of the pose estimation to see how each of the optimizations impacts them. Unless explicitly stater otherwise all the CPU test were run on a single thread and the measured times ware:

Depth processing - all the processing of the depth map including:

- hole filling (Section 4.4)

Figure 20: Frame Sequence 1. Note the distortion of the models caused by missing values and re-rasterization with grid size of 2.5mm.

- filtering (Section 4.3)
- rasterization (Section 4.2)

Execution time is dependent on the size of the input depth map and settings of the procedure itself, such as number of filtering iterations and raster size.

PPF creation - creation of the PPF from the point pair. Described in Sections 3.1 and 4.6. This operation is relatively trivial. Its execution time can be affected mostly by micro optimizations, such as using fast but less accurate approximation of the inverse cosine.

PPF search - search of the PPF in the hash map.

Pose voting - addition of the votes into the accumulator. Execution time depends on the number of votes a single PPF has to add into the accumulator, which depends on the global model description generated in the offline phase.

Accumulator processing - finding of the pose with the most votes. Execution time depends only on the accumulator size, which depends on the number of points in the reference

model and angle discretization. Both of these parameters are determined in the offline phase.

Our OpenCL implementation performs **PPF creation**, **PPF search** and **Pose voting** inside a single kernel. Therefore, we cannot measure the times of these segments separately. For that reason we call the aggregation of these segments **PPF Processing** and measure its kernel execution time. We also measure kernel execution time of the **Accumulator processing**.

We will also have to evaluate the impact of the optimization on the accuracy of the pose estimation, for that we need a methodology to determine if the pose was estimated correctly. Our methodology for that is to generate the ground truths with out application set to higher than normal precision, and then perform visual confirmation of the found poses, and after visually confirming that all the found poses were reasonably correct, we accept them as ground truth. Due to this impreciseness of the ground truth we have decided to use a simple binary metric of pose either being reasonably close to the ground truth or not.

Restricting Point Pair Features

Restricting of the Point Pair Features should lead to a smaller number of PPFs in the global model description. Therefore, PPF search may be faster, but our implementation uses hash map for the search, due to this we do not expect a significant change in the PPF search time.

We do expect more significant change in the pose voting time after the removal of the PPFs with low angle between point normals $\angle(n_1, n_2)$, because such PPFs place large number of pose votes.

Experiment 1

In our first experiment we used frame sequence 1. To ensure repeatability of the experiment we also used constant method of selection of points for the pose estimation and carried out experiment with different sizes of point skips.

Because our PPF restriction is based on restricting the angle between point normals $\angle(n_1, n_2)$, he have decided on this naming convention for the used angle restrictions:

none - no restriction.

high - PPFs with angle $\angle(n_1, n_2)$ higher than 105 degrees rejected.

high and low - PPFs with angle $\angle(n_1, n_2)$ lower than 5 degrees and higher than 105 degrees rejected.

This experiment was performed with CPU calculator, and therefore we have decided to present more detailed results. Tables 1 2 3 4 5 show results of the experiments with different PPF angle restrictions with skips of 2, 3, 6, 9 and 12 points respectively.

PPF Angle Restrictions	Correct Poses	Total Time [s]	PPF Creation Time [s]	PPF Search Time [s]	Pose Voting Time [s]	Accum. Processing Time [s]
none	360/360	78.203	22.131	15.609	14.665	14.755
high	360/360	77.464	22.112	15.353	14.250	14.751
high and low	360/360	66.468	22.036	15.242	3.331	14.891

Table 1: Restricting point pair features, experiment 1, point skips: 2. Changes in PPF angle restrictions should have impact on PPF search time and pose voting time. Removal of PPFs with high angle $\angle(n_1, n_2)$ should mostly impact PPF search time because such PPFs are rarely in the scenes, and therefore their pose voting will not occur very frequently. Removal of PPFs with low angle $\angle(n_1, n_2)$ should have significant impact on pose voting time because such PPFs are found in every plane in the scene. Therefore, their removal will also most likely have a negative impact on the accuracy of the pose estimation.

Angle Restrictions	Correct Poses	Total Time [s]	PPF Creation Time [s]	PPF Search Time [s]	Pose Voting Time [s]	Accum. Processing Time [s]
none	360/360	35.862	9.791	7.103	6.662	6.545
high	360/360	35.332	9.777	6.867	6.437	6.556
high and low	360/360	30.309	9.721	6.781	1.486	6.586

Table 2: Restricting point pair features, experiment 1, point skips: 3

Angle Restrictions	Correct Poses	Total Time [s]	PPF Creation Time [s]	PPF Search Time [s]	Pose Voting Time [s]	Accum. Processing Time [s]
none	341/360	10.640	2.574	1.905	1.736	1.741
high	342/360	10.545	2.580	1.891	1.685	1.751
high and low	342/360	9.268	2.566	1.878	0.402	1.756

Table 3: Restricting point pair features, experiment 1, point skips: 6

Angle Restrictions	Correct Poses	Total Time [s]	PPF Creation Time [s]	PPF Search Time [s]	Pose Voting Time [s]	Accum. Processing Time [s]
none	270/360	5.803	1.166	0.885	0.834	0.804
high	270/360	5.690	1.161	0.863	0.804	0.800
high and low	265/360	5.068	1.155	0.858	0.201	0.810

Table 4: Restricting point pair features, experiment 1, point skips: 9

Angle Restrictions	Correct Poses	Total Time [s]	PPF Creation Time [s]	PPF Search Time [s]	Pose Voting Time [s]	Accum. Processing Time [s]
none	159/360	4.023	0.658	0.490	0.532	0.452
high	161/360	3.963	0.656	0.489	0.515	0.447
high and low	165/360	3.579	0.657	0.488	0.117	0.458

Table 5: Restricting point pair features, experiment 1, point skips: 12

Experiment Conclusion

Removal of PPFs with high angle $\angle(n_1, n_2)$ did not caused any degradation in the accuracy of the pose estimation. However, it also did not cause a significant time improvement, this is most likely, because occurrence of such PPFs in the scenes is very rare, and therefore their removal mainly decreased the size of the hash table used during the PPF search.

Removal of PPFs with high and low angle $\angle(n_1, n_2)$ reduced the pose estimation time by at least 10% and also reduced the accuracy of pose estimation in the experiments with points skips of 9 and 16. However, accuracy of the unrestricted PPFs in these experiment was sub-optimal for such trivial test scene. For that reason we consider it a reasonable trade-off. Most of the time reduction occurred in pose voting, as expected.

Times of the PPF creation and accumulator processing for different PPF angle restrictions, but same point skips were withing the margin of error. This supports the fact that restriction of PPFs has no impact on there operations.

Experiment 2

Our second experiment used the same methodology and frame sequence as our previous experiment. However, OpenCL devices were used for the pose estimation.

The following GPUs were used as OpenCL devices:

- GeForce GT 820M, Table 6
- GeForce GTX 1070, Table 7

Experiment Conclusion

As in the previous experiment the removal of PPFs with high angle $\angle(n_1, n_2)$ did not cause a significant time improvement. However, removal of PPFs with high and low angle $\angle(n_1, n_2)$ caused the total time to decreased by approximately 40-50% with GeForce GT 820M as OpenCl device and 38-42% with GeForce GTX 1070, without any reduction of accuracy.

Accumulator processing times for different PPF angle restrictions, but same point skips were again withing the margin of error. Because restriction of the PPFs has no impact on them.

Point Skips	Angle Restrictions	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
1	none	360/360	72.380	1.394	46.526	22.657
	high	360/360	72.437	1.416	46.692	22.558
	high and low	360/360	35.968	1.407	10.156	22.646
2	none	360/360	20.211	1.401	11.925	5.913
	high	360/360	20.214	1.379	11.964	6.002
	high and low	360/360	10.764	1.427	2.622	5.735
3	none	360/360	10.564	1.422	5.584	2.841
	high	360/360	10.548	1.441	5.585	2.842
	high and low	360/360	5.889	1.428	1.196	2.497
4	none	360/360	7.137	1.434	3.441	1.694
	high	360/360	7.190	1.464	3.492	1.681
	high and low	360/360	4.186	1.447	0.696	1.406

Table 6: Restricting point pair features, experiment 2, GeForce GT 820M. Changes in PPF angle restrictions should have same impact as in the previous experiment (Table 1) but PPF search and pose voting are part of a single OpenCL kernel. Therefore, their aggregate time is referent to as PPF Processing Time.

Point Skips	Angle Restrictions	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
1	none	360/360	11.231	0.479	5.372	3.105
	high	360/360	11.196	0.479	5.354	3.108
	high and low	360/360	6.918	0.482	0.962	3.104
2	none	360/360	3.420	0.477	1.541	0.785
	high	360/360	3.405	0.476	1.548	0.778
	high and low	360/360	2.138	0.478	0.270	0.767
3	none	360/360	2.334	0.476	1.118	0.344
	high	360/360	2.325	0.476	1.113	0.344
	high and low	360/360	1.371	0.475	0.161	0.327
4	none	360/360	2.006	0.474	1.000	0.195
	high	360/360	2.031	0.473	1.027	0.193
	high and low	360/360	1.150	0.474	0.129	0.187

Table 7: Restricting point pair features, experiment 2, GeForce GTX 1070

We assume that the reason for much more significant time improvement is because additions to the accumulator requires atomic random access into global memory of the OpenCL devices. Which is one of the slowest memory operations on such devices.

Lower total time improvement on the GeForce GTX 1070 was caused by PPF processing taking less of the overall time on this device.

Experiment 3

The aim of this experiment is to show, that restricting of the PPFs will not lead to significant time improvements in the situations when most of the PPFs in the scene will not be found in the global model description. Therefore, pose voting will not take significant amount of the overall time. For this reason we used frame sequence 2 in which the sought object takes much less space and has different color than the rest of the scene. The rest of the testing methodology remains unchanged.

The following GPUs were used as OpenCL devices:

- GeForce GT 820M, Table 8
- GeForce GTX 1070, Table 9

Point Skips	**Angle Restrictions**	**Correct Poses**	**Total Time [s]**	**Depth Processing Time [s]**	**PPF Processing Time [s]**	**Accum. Processing Time [s]**
2	none	359/360	220.521	2.565	37.124	172.617
	high	359/360	217.951	2.525	34.680	172.557
	high and low	359/360	212.316	2.380	29.337	172.247
3	none	359/360	100.292	2.459	16.591	77.053
	high	359/360	99.116	2.607	15.467	76.853
	high and low	356/360	96.431	2.408	13.068	76.802
4	none	346/360	57.922	2.654	9.266	43.378
	high	346/360	57.279	2.625	8.645	43.349
	high and low	346/360	55.737	2.419	7.312	43.422
5	none	322/360	39.322	2.401	6.357	28.547
	high	320/360	38.538	2.444	5.963	28.231
	high and low	317/360	37.292	2.477	4.770	28.152
6	none	261/360	27.871	2.397	4.334	19.367
	high	261/360	27.798	2.349	4.069	19.614
	high and low	256/360	27.086	2.365	3.278	19.901

Table 8: Restricting point pair features, experiment 3, GeForce GT 820M

Experiment Conclusion

As in the previous experiments removal of PPFs with high angle $\angle(n_1, n_2)$ did not cause a significant time improvement. And as expected, removal of PPFs with high and low angle $\angle(n_1, n_2)$ caused much lower time improvement than in the previous experiment. Total time was decreased by approximately 3-5% with GeForce GT 820M and 6-10% with GeForce GTX 1070. Removal of PPFs with high and low angle $\angle(n_1, n_2)$ also lead to a slight degradation of accuracy. As mentioned in the experiment description, we have expected such results and the next experiment will evaluate a possible solution for this type of scene sequences.

Point Skips	Angle Restrictions	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
2	none	359/360	37.652	0.594	4.995	23.010
	high	359/360	37.337	0.592	4.749	22.991
	high and low	359/360	35.047	0.614	2.293	23.057
3	none	359/360	17.632	0.586	2.424	10.339
	high	359/360	17.508	0.585	2.319	10.324
	high and low	356/360	16.580	0.607	1.093	10.565
4	none	346/360	10.683	0.579	1.724	5.892
	high	346/360	10.626	0.576	1.659	5.887
	high and low	346/360	9.855	0.617	0.716	5.967
5	none	322/360	7.682	0.575	1.582	3.771
	high	320/360	7.613	0.575	1.527	3.760
	high and low	317/360	6.723	0.619	0.539	3.770
6	none	261/360	5.659	0.577	1.067	2.604
	high	261/360	5.581	0.575	1.027	2.594
	high and low	256/360	5.063	0.617	0.393	2.615

Table 9: Restricting point pair features, experiment 3, GeForce GTX 1070

Experiment 4

The aim of this experiment is to show, that restricting of the PPFs may lead to significant time improvement even in the situations when most of the PPFs in the scene will not be found in the global model description, but only if the adaptive selection of the points for the pose estimation is used. We expect such results, because adaptive selection of the points for the pose estimation should mostly select only the points, that are part of the sough model, and therefore remove the problem of most PPFs not being found in the global model description.

We configured point selection to select 250 points during adaptive selection and use 3 point skips for the first frame and as a fall-back for the frames where previously estimated pose and pose estimated before it were too different.

However, unlike in our previous experiments, the pose estimation may fail in different frames. This is cause by the random distribution of the adaptively selected points. Due to this we carried out 30 test for each test case, and the tables present the averages and standard deviations obtained form these tests.

Because removal of PPFs with high angle $\angle(n_1, n_2)$ did not cause any significant time improvement in the previous experiments, we have decided to omit it from this experiment.

The following GPUs were used as OpenCL devices:

- GeForce GT 820M, Tables 10, 11
- GeForce GTX 1070, Tables 12, 13

Angle Restrictions	Average Correct Poses	Standard Deviation	Average Total Time [s]	Standard Deviation [s]
none	357.4	1.57	22.15	0.28
high and low	358.1	1.63	17.90	0.30

Table 10: Restricting point pair features, experiment 4, GeForce GT 820M, 1/2

Angle Restrictions	Average PPF Processing Time [s]	Standard Deviation [s]	Average Accum. Processing Time [s]	Standard Deviation [s]
none	9.901	0.050	8.720	0.232
high and low	5.645	0.033	8.765	0.258

Table 11: Restricting point pair features, experiment 4, GeForce GT 820M, 2/2

Angle Restrictions	Average Correct Poses	Standard Deviation	Average Total Time [s]	Standard Deviation [s]
none	357.8	1.29	4.59	0.12
high and low	358.0	1.12	3.48	0.10

Table 12: Restricting point pair features, experiment 4, GeForce GTX 1070, 1/2

Angle Restrictions	Average PPF Processing Time [s]	Standard Deviation [s]	Average Accum. Processing Time [s]	Standard Deviation [s]
none	1.684	0.016	1.323	0.077
high and low	0.607	0.006	1.370	0.071

Table 13: Restricting point pair features, experiment 4, GeForce GTX 1070, 2/2

Experiment Conclusion

Accuracy of the pose estimation was not impacted by the removal of the PPFs. Their averages do differ, but their standard deviations are much higher than the difference of their averages. Therefore, we do not consider the difference to be statistically significant.

Removal of PPFs with high and low angle $\angle(n_1, n_2)$ caused the total average time to decrease by approximately 19% with GeForce GT 820M as OpenCL device. If we multiply standard deviations of the total times by three, and then subtract the one for the total time without PPF angle restrictions from the average total time without PPF angle restrictions, and add the one for the total time with PPF angle restrictions to the average total time with the PPF angle restrictions, and calculate time reduction based on these values. We end up with time reduction by approximately 11%. This result represent the worst case scenario. If we swap the addition

and subtraction of the three standard deviations to obtain the best case scenario, we end up with time reduction by approximately 26%. Due to this, we consider the time improvement to be statistically significant.

Removal of PPFs with high and low angle $\angle(n_1, n_2)$ caused the total average time to decrease by approximately 24% with GeForce GTX 1070 as OpenCL device. If we multiply standard deviations of the total times by three, and then subtract the one for the total time without PPF angle restrictions from the average total time without PPF angle restrictions, and add the one for the total time with PPF angle restrictions to the average total time with the PPF angle restrictions, and calculate time reduction based on these values. We end up with time reduction by approximately 10%. This result represent the worst case scenario. If we swap the addition and subtraction of the three standard deviations to obtain the best case scenario, we end up with time reduction by approximately 35%. Due to this, we consider the time improvement to be statistically significant.

Reference Model Culling

Reference model culling should lead to smaller number of points in the global model description, and therefore it should impact pose voting and accumulator processing times. Pose voting time was already reduced with restricting of the PPFs, which will be used in all following experiments. Further impact of the reference model culling on it is hard to estimate.

Impact of the reference model culling on the accumulator processing time is easier to estimate, because accumulator processing time is composed of two phase search for the best value in the accumulator. First phase searches for the maximal value in each row of the accumulator. Therefore, if we assume, that culled reference model has approximately half of the points, the accumulator will have approximately half of the rows. Because of that, we expect the time of this phase to decrease approximately by half. Second phase searches for the maximal value in the maximal values from the first phase. Therefore, we should obtain the same time improvement.

Experiment 1

In this experiment we used frame sequence 1. Also results of the experiment in the Section 5.3.3 are used for the comparison with the values obtained in this experiment. Original reference model has 1500 points. Minimum and maximum of points in culled reference models is 280 and 647 respectively, which is around 19% and 43% of the original point count. Therefore, we expect accumulator processing to be decreased by approximately 60%.

The following GPUs were used as OpenCL devices:

- GeForce GT 820M, Table 14
- GeForce GTX 1070, Table 15

Point Skips	Model Culling	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
1	no	360/360	35.968	1.407	10.156	22.646
	yes	360/360	15.720	1.451	5.475	6.936
2	no	360/360	10.764	1.427	2.622	5.735
	yes	360/360	5.480	1.485	1.416	1.738
3	no	360/360	5.889	1.428	1.196	2.497
	yes	357/360	3.485	1.461	0.645	0.782
4	no	360/360	4.186	1.447	0.696	1.406
	yes	351/360	2.808	1.467	0.382	0.447

Table 14: Reference model culling, experiment 1, GeForce GT 820M. Reference model culling shroud decrease the time of PPF processing and accumulator processing.

Point Skips	Model Culling	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
1	no	360/360	6.918	0.482	0.962	3.104
	yes	360/360	3.945	0.477	0.276	0.922
2	no	360/360	2.138	0.478	0.270	0.767
	yes	360/360	1.422	0.482	0.080	0.233
3	no	360/360	1.371	0.475	0.161	0.327
	yes	357/360	1.046	0.478	0.059	0.105
4	no	360/360	1.150	0.474	0.129	0.187
	yes	351/360	0.962	0.487	0.057	0.065

Table 15: Reference model culling, experiment 1, GeForce GTX 1070

Experiment Conclusion

Accumulator processing time was decreased by approximately 70-65% for both OpenCL devices. Accuracy of the pose estimations was slightly impacted by the culling, but we consider it a reasonable trade-off.

Experiment 2

In this experiment we used frame sequence 2. Also results of the experiment in the Section 5.3.5 are used for the comparison with the values obtained in this experiment. Original reference model has 2100 points. Minimum and maximum of points in culled reference models is 570 and 1040 respectively, which is around 35% and 70% of the original point count. Therefore, we expect accumulator processing to be decreased by approximately 40%.

The following GPUs were used as OpenCL devices:

- GeForce GT 820M, Table 16
- GeForce GTX 1070, Table 17

Point Skips	Model Culling	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
2	no	359/360	212.316	2.380	29.337	172.247
	yes	359/360	107.414	2.517	23.493	74.530
3	no	356/360	96.431	2.408	13.068	76.802
	yes	357/360	50.503	2.343	10.556	33.505
4	no	346/360	55.737	2.419	7.312	43.422
	yes	341/360	32.551	2.656	6.087	21.125
5	no	317/360	37.292	2.477	4.770	28.152
	yes	318/360	23.859	2.516	4.032	15.175
6	no	256/360	27.086	2.365	3.278	19.901
	yes	250/360	19.689	2.467	2.881	12.482

Table 16: Reference model culling, experiment 2, GeForce GT 820M

Point Skips	Model Culling	Correct Poses	Total Time [s]	Depth Processing Time [s]	PPF Processing Time [s]	Accum. Processing Time [s]
2	no	359/360	35.047	0.614	2.293	23.057
	yes	359/360	22.699	0.618	1.594	11.554
3	no	356/360	16.580	0.607	1.093	10.565
	yes	356/360	10.978	0.620	0.760	5.283
4	no	346/360	9.855	0.617	0.716	5.967
	yes	341/360	6.715	0.619	0.479	3.067
5	no	317/360	6.723	0.619	0.539	3.770
	yes	318/360	4.883	0.612	0.356	2.074
6	no	256/360	5.063	0.617	0.393	2.615
	yes	250/360	4.053	0.622	0.290	1.690

Table 17: Reference model culling, experiment 2, GeForce GTX 1070

Experiment Conclusion

Accumulator processing time was decreased by approximately 50-35% for GeForce GTX 1070, and 56-37% for GeForce GT 820M. Accuracy of the pose estimations was slightly impacted by the culling, but we consider it a reasonable trade-off.

Parallelization

The aim of this experiment is to show the scalability of our CPU parallelization, and to show the performance improvement gained by the usage of OpenCL devices. In this experiment we used frame sequence 2. All previously mentioned optimizations were used during the pose estimations. Table 18 contains results of the CPU parallelization. The results were obtained on a system with AMD FX(tm)-8150 Eight-Core Processor and GeForce GTX 1070 but only CPU was used for the pose estimation. Each value represents only a single run of our application because we believe that the time each run took is relatively long. Therefore, time improvements caused by the usage of more threads are significant enough to use there single run results at least as an approximation of the scalability of our implementation.

Table 19 shows averages and standard deviations obtained from 30 runs of our applications with GeForce GTX 1070 as OpenCL device used for the pose estimation, and various number of CPU threads.

Threads	**Correct Poses**	**Total Time [s]**	**Total Time [%]**
1	354/360	191.22	100.00
2	358/360	97.51	50.99
3	354/360	59.99	31.37
4	357/360	46.78	24.47
5	360/360	37.53	19.63
6	358/360	30.93	16.17
7	359/360	27.97	14.63
8	359/360	26.44	13.82

Table 18: Parallelization, AMD FX(tm)-8150 Eight-Core Processor. Result with single thread is used as baseline for the percentages.

CPU threads	**Average Correct Poses**	**Standard Deviation**	**Average PPF Processing Time [s]**	**Standard Deviation [s]**
0	357.87	1.11	2.474	0.035
4	356.97	1.97	2.431	0.027
8	357.77	1.36	2.479	0.030

Table 19: Parallelization GeForce GTX 1070, and AMD FX(tm)-8150 Eight-Core Processor. Note that GeForce GTX 1070 is around 77 times faster than single core from the Table 18. This is most likely the reason why adding 4 or 8 thread to it had no statistically significant inpact on the overall calculation time.

Experiment Conclusion

The percentages in the Table 18 show almost perfect scaling with the number of threads used for the pose estimation. Some of them are slightly better than they can theoretically be. However, as we have mentioned before these results are from single run, and therefore should be considered as more of a rough estimates.

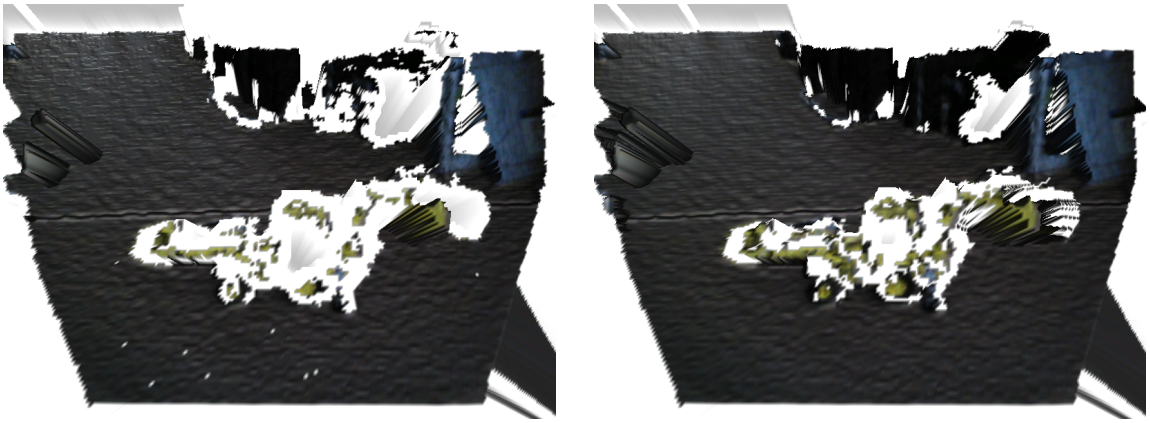
If we compare the total time of single CPU thread from the Table 18 with the average total time of GeForce GTX 1070 with no additional CPU threads from the Table 19. We will found out that the calculation with GeForce GTX 1070 was proximately 77 times faster than the single CPU thread. This explains why the data in the Table 19 suggest that the addition of 4 and 8 CPU thread had no impact on the calculation time.

Real Data

During our testing we had access to Orbbec Astra and Orbbec Astra Pro depth cameras. Although our sought models in our synthetically created scenes are real world object, we were unable to use them in this experiments. We do not have mechanical part used in frame sequence 2. Despise the fact that we have LEGO 6876 Alienator, we were unable to capture it because its black glossy surfaces are undetectable by the depth cameras. Fortunately, we had access to LEGO 8862 Backhoe Grader which is mostly composed of yellow LEGO bricks. Figure 21 shows initial reference model that was used for the pose estimation.

Figure 21: LEGO 8862 Backhoe Grader, initial reference model

Our first RGB-D caption of LEGO 8862 Backhoe Grader was made with Orbbec Astra depth camera. Figure 22a shows the captured scene. As you can see Orbbec Astra failed to capture most of the model making this scene not usable for our purposes. We decided to take take multiple shots of this scene and merged them to improve the quality of the captured scene. Figure 22b shows the result of 134 merged frames the improvement is visible but not good enough for our primary depth based method. This lead us to switch to Orbbec Astra Pro depth camera, and start with a merge of multiple frames to get better result. Figure 23 shows the result of 154 merged frames without color because we were unable to capture it. In our opinion, this scene was sufficient for our purposes.



(a) Single frame (b) Merge of 134 frames

Figure 22: LEGO 8862 Backhoe Grader scene, Orbbec Astra

Unfortunately, our initial results were not favorable. Even after using only 32 angle steps and very high PPF angle restrictions. We were only able to estimate the correct pose of the sought model if we restricted the search space to the space where the model was. We suspected that the reason for our poor results is the fact that our reference model contains too many small details, and therefore does not correspond to the actual model captured by the depth camera. To test this hypothesis we created new simplified reference model by removing the small LEGO bricks and converting most of the bigger ones to their convex hulls, and then used combination of decimate and remesh modifiers in Blender to make it look more like a model captured by the depth camera. Figure 24 shows the modified model. The model is missing its back dinging arm because it is articulated, and we had mispositioned it in our real world object. Therefore, we decided to remove it from our model reference instead of trying to find its correct articulation. With this updated model reference we were able to estimate its pose in the test scene. Figure 25 shows the estimated pose.

To give you some information about the robustness of the estimated pose, we had collected the data from pose clustering. Table 20 shows the number of different clusters with the same number of pose estimations supporting them. It also shows how many of such clusters contains correct poses. As correct poses we mean poses that have the distance from the estimated pose

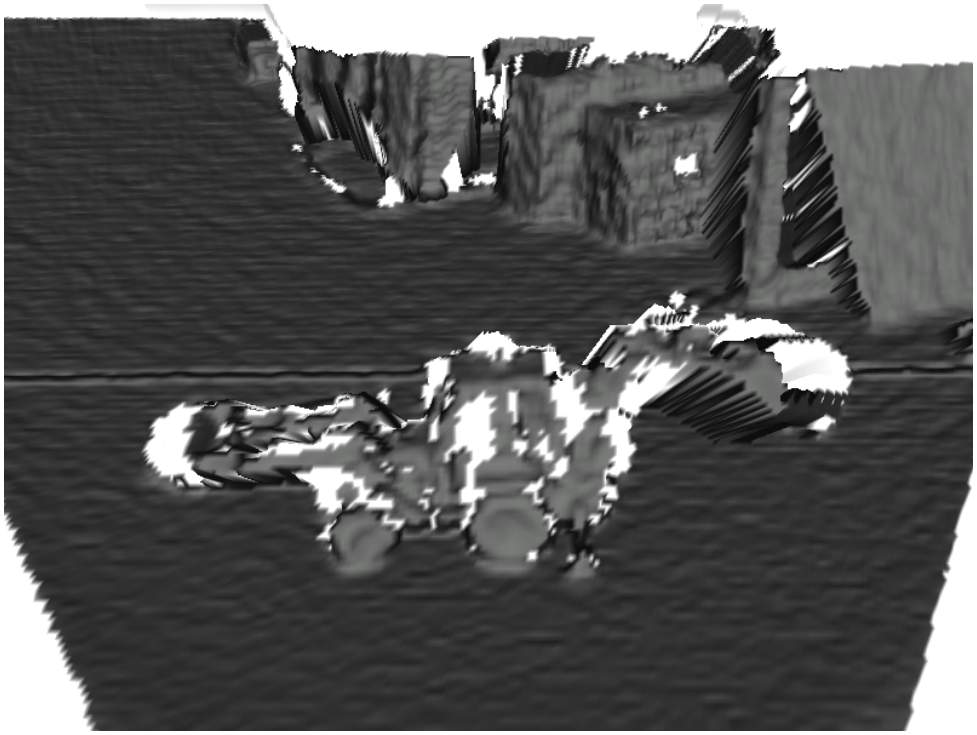


Figure 23: LEGO 8862 Backhoe Grader scene, Orbbec Astra Pro, merge of 154 frames

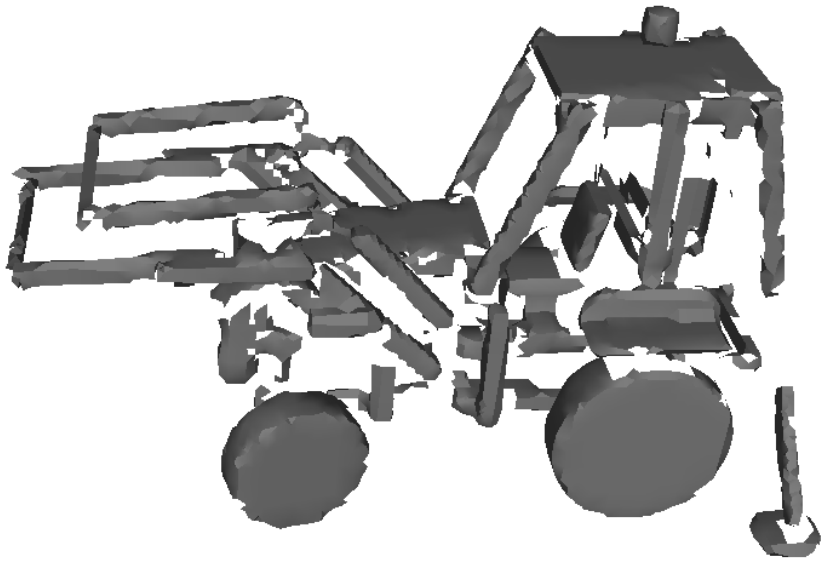


Figure 24: LEGO 8862 Backhoe Grader, updated reference model

Figure 25: LEGO 8862 Backhoe Grader scene with pose estimation, Orbbec Astra Pro, merge of 154 frames. Note the misaligned back dinging arm.

lower than 2cm, and highest angle difference of 14 degrees. Estimated pose is the pose in the first cluster with the highest number of pose estimations supporting it. Table 20 shows that clusters with 9, 10, 13 and 14 poses supporting them all contain only correct poses. This can be view as a measure of a robustness be it scene specific one.

We also used adaptive point selection on this single frame. We are aware that it is not ideal to use single frame as frame sequence but we wanted to show that usage of adaptive point selection will increase the range of clusters with only correct poses. However, because of the random nature of the adaptive point selection each of the result will be different. Therefore we randomly picked three of them and represented them by Tables 21, 22 and 23. All of the tables show clusters with higher support than initial estimation (Table 20).

After we had succeed to estimate the pose in the merge of 154 the frames from Orbbec Astra Pro, we have decided to try to estimate pose from single captured frame. We were able to estimate the pose of the sought object in that scene. Figure 26 shows captured scene, and the same scene with the sough object with correctly estimated pose.

To give you some information about the robustness of the estimated pose, we had collected the data from pose clustering. Table 24shows the number of different clusters with the same number of pose estimations supporting them. Table shows that clusters with 9 pose estimations supporting them contain only correct poses. This Can be interpreted as less robust result compared to the results obtained from the merge of 154 frames, as expected. We had also tested

Poses in Cluster	Cluster Count	Correct Cluster Count
1	83910	12
2	4196	2
3	410	6
4	86	0
5	37	2
6	5	2
7	2	2
8	1	0
9	1	1
10	1	1
13	1	1
14	1	1

Table 20: Pose clusters, Orbbec Astra Pro, merge of 154 frames. First row shows that there exist 83910 clusters with 1 pose estimation which supports them, and 12 of them contain pose that is close enough to the correct pose. The correct pose is the pose in the 1st cluster with the most of the pose estimations supporting it. In this case it is 14. This implies that if the last row of the table contains more clusters than correct clusters, then we are unable to distinguish if there are multiple validly detected object, or no valid object at all. In this case clusters with 9, 10, 13 and 14 poses supporting them all contain only correct poses. This can be view as a measure of a robustness be it scene specific one.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	3906	20
2	214	5
3	25	5
4	17	5
5	4	4
7	1	1
10	1	1
11	1	1
16	1	1
20	1	1
21	1	1
24	1	1

Table 21: Pose clusters, Orbbec Astra Pro, merge of 154 frames, adaptive point selection 1/3. Note that clusters with 5, 7, 10, 11, 16, 20, 21 and 24 poses supporting them all contain only correct poses which is improvement compared to the Table 20.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	3935	8
2	207	3
3	25	0
4	7	0
5	2	0
6	2	0
7	2	2
9	3	3
10	2	2
11	1	1
13	1	1
15	1	1
16	1	1
17	1	1

Table 22: Pose clusters, Orbbec Astra Pro, merge of 154 frames, adaptive point selection 2/3. Note that clusters with 7, 9, 10, 11, 13, 15, 16 and 17 poses supporting them all contain only correct poses which is improvement compared to the Table 20.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	3929	14
2	200	2
3	27	8
4	9	5
5	6	4
6	1	0
7	2	2
8	1	1
14	1	1
18	1	1
19	1	1
23	1	1

Table 23: Pose clusters, Orbbec Astra Pro, merge of 154 frames, adaptive point selection 3/3. Note that clusters with 7, 8, 14, 18, 19 and 23 poses supporting them all contain only correct poses which is improvement compared to the Table 20.

the pose estimation with adaptive point selection. We again randomly picked three clustering results and represented them by Tables 25, 26 and 27. All of the tables show clusters with higher support than initial estimation.

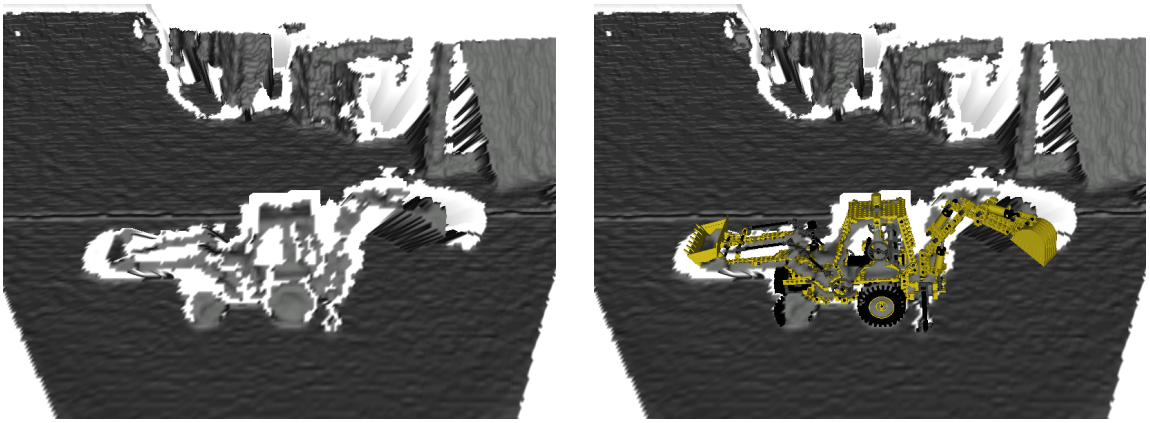


Figure 26: EGO 8862 Backhoe Grader scene with pose estimation, Orbbec Astra Pro, single frame. Note significantly more missing values in comparison with the Figure 23.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	76935	22
2	3664	3
3	336	5
4	49	1
5	17	1
6	4	0
9	2	2

Table 24: Pose clusters, Orbbec Astra Pro, single frame. Only Clusters with 9 poses supporting them contain only correct poses. This can be seen as less robust result than the one obtained from the merge of 154 frames in the Table 20.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	3855	8
2	249	5
3	21	2
4	5	1
5	3	3
6	1	1
7	2	2
8	1	1
11	1	1
13	3	3

Table 25: Pose clusters, Orbbec Astra Pro, single frames, adaptive point selection 1/3. Note that clusters with 5, 6, 7, 8, 11 and 13 poses supporting them all contain only correct poses which is improvement compared to the Table 24.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	3871	18
2	232	8
3	18	0
4	1	0
5	8	4
7	4	4
8	2	2
11	4	4

Table 26: Pose clusters, Orbbec Astra Pro, single frames, adaptive point selection 2/3. Note that clusters with 7, 8 and 11 poses supporting them all contain only correct poses which is improvement compared to the Table 24.

Poses in Cluster	Cluster Count	Correct Cluster Count
1	3954	16
2	177	1
3	25	3
4	8	2
5	1	1
6	2	2
10	2	2
12	1	1
15	1	1
16	1	1
17	1	1

Table 27: Pose clusters, Orbbec Astra Pro, single frames, adaptive point selection 3/3. Note that clusters with 5, 6, 10, 12, 15, 16 and 17 poses supporting them all contain only correct poses which is improvement compared to the Table 24.

Conclusion

We have succeeded in implementing a method for the pose estimation of a known object inside a scene. Our method was inspired by the [10]. The result of our implementation is a multi-platform C++ console application that uses point cloud of the sought object, 3D model of the sought object for visualization and RGB-D image of the scene as it inputs and displays the estimated pose by rendering the 3D model of the sought object into the scene. It also displays the transformation matrix of the pose and the time pose estimation took.

We have carried out numerous experiments to evaluate the impact of various optimizations of our method on the calculation time and the accuracy of the pose estimation:

Restricting point pair features : Four different experiments were carried out to properly estimate the impact of PPF restriction (Section 5.3). The obtained results show that it can decrease the calculation time by at least 10% in the worst case scenario when used correctly. Best case scenario shows up to 50% time reduction. The experiment with somewhat realistic use case showed the time reduction of roughly 20-25%. Accuracy of the pose estimation is significantly affected by the PPF restriction only when it was undesirable to begin with.

Reference model culling : Two experiments were carried out the estimate the impact of reference model culling. The obtained results show that it can decrease the calculation time by up to 70% in the best case scenario. More realistic scenarios showed the time decrease by 50-35%. Accuracy of the pose estimation was not significantly affected by the addition of reference model culling.

Parallelization : Table 18 shows that our CPU parallelization was able of almost perfect scaling on the AMD FX(tm)-8150 Eight-Core Processor. Pose estimation of 360 frame long frame sequence took 26.44s with 8 threads. This gives us average of 74ms per frame. The usage of GeForce GTX 1070 as OpenCL device (Table 19) gave us the time of 2.474s for the same frame sequence sequence. This time is roughly 77 times faster than single thread time.

We were able to successfully estimate the pose of 3 different object in synthetically generated and real scenes. Below is a brief summary that discuses the results we have obtained in each scene:

Frame sequence 1 : We never used this scene for direct time measurement because of its simplistic nature. However, Table 15 shows that GeForce GTX 1070 was able to correctly estimate all the poses in the frame sequence with an average time of 4ms per frame.

Frame sequence 2 : Most realistic test of this sequence was performed in the parallelization experiments (Section 5.5). Table 19 shows that GeForce GTX 1070 was able to reach

average of 357.87 correct poses from 360 total poses with an average time of 7ms per frame.

Real data : We were unable to estimate the pose of the sought object in the scenes captured by the Orbbec Astra (Figures 22b, 22a) because most of the depth values were missing which is problematic for our depth based method. However, we were able to successfully estimate the pose from the scenes obtained by Orbbec Astra Pro (Figures 25, 26).

We are relatively pleased with the results we were able to obtain. However, there are also areas in which further research may improve our application.

Depth preprocessing : Most of our experiments were carried out on the synthetically generated scenes. Therefore, out implementation of depth enhancement was not properly tested.

Clustering algorithm : Our implementation uses simple clustering that puts poses that are close to each other into the same cluster and averages the pose in that cluster. It is dependent on the order in which poses are added into into clusters which makes clusters unstable when multiple calculators work on the same scene. We select first cluster with highest number of poses as the best one but this makes this clustering unsuitable for detection of multiple identical object with different poses. Possible better clustering method may improve the stability of the results, allowing for multiple object detection and possibly enhancing robustness of the pose detection.

Rough object pose estimation : Our application checks the entire scene for the sought object with a method that is used for precise object pose estimation. This might be seen as wasteful. Therefore, implementation of methods that put constrains on the area of pose estimation may improve the speed of the pose estimation. Adaptive selection of points for the pose estimation partially does this but it can only work with frame sequences. It also assumes that the pose of the sought object does not change much between the frames.

References

[1] Cg 3.1 toolkit documentation cg / standard library / acos. `http://developer.download.nvidia.com/cg/acos.html`. Accessed: 2018-08-04.

[2] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, ninth dover printing, tenth gpo printing edition, 1964.

[3] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze. Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In *2013 IEEE International Conference on Robotics and Automation*, pages 2104–2111, May 2013.

[4] Paul J. Besl and Ramesh C. Jain. Three-dimensional object recognition. *ACM Comput. Surv.*, 17(1):75–145, March 1985.

[5] Tolga Birdal and Slobodan Ilic. Point pair features based object detection and pose estimation revisited. In *Proceedings of the 2015 International Conference on 3D Vision*, 3DV '15, pages 527–535, Washington, DC, USA, 2015. IEEE Computer Society.

[6] Richard J. Campbell and Patrick J. Flynn. A survey of free-form object representation and recognition techniques. *Comput. Vis. Image Underst.*, 81(2):166–210, February 2001.

[7] C. Choi and H. I. Christensen. 3d pose estimation of daily objects using an rgb-d camera. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3342–3349, Oct 2012.

[8] Changhyun Choi and Henrik I. Christensen. Rgb-d object pose estimation in unstructured environments. *Robot. Auton. Syst.*, 75(PB):595–613, January 2016.

[9] Chin Seng Chua and Ray Jarvis. Point signatures: A new representation for 3d object recognition. *Int. J. Comput. Vision*, 25(1):63–85, October 1997.

[10] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 998–1005. Ieee, 2010.

[11] M. Germann, M. D. Breitenstein, I. K. Park, and H. Pfister. Automatic pose estimation for range images on the gpu. In *Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM 2007)*, pages 81–90, Aug 2007.

[12] T. Hodaň, X. Zabulis, M. Lourakis, Š. Obdržálek, and J. Matas. Detection and fine 3d pose estimation of texture-less objects in rgb-d images. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4421–4428, Sept 2015.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. 06 2014.

[14] Andrew Johnson and Martial Hebert. Using spin-images for efficient object recognition in cluttered 3-d scenes. 08 1998.

[15] In Kyu Park, Marcel Germann, Michael Breitenstein, and Hanspeter Pfister. Fast and automatic object pose estimation for range images on the gpu. 21:749–766, 08 2010.

[16] Ilya Lysenkov, Victor Eruhimov, and Gary Bradski. Recognition and pose estimation of rigid transparent objects with a kinect sensor. In *in Proc. of Robotics: Science and Systems*, 2012.

[17] Tahir Rabbani and Frank Van Den Heuvel. Efficient hough transform for automatic detection of cylinders in point clouds. In *In: IAPRS, XXXVI, 3/W19*, pages 60–65, 2005.

[18] M. Schwarz, H. Schulz, and S. Behnke. Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1329–1335, May 2015.

[19] F. Stein and G. Medioni. Structural indexing: efficient 3-d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):125–145, Feb 1992.

[20] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. 11 2017.

Appendix on CD

The attached CD contains:

sources This folder contains implementation of the C++ application for the pose estimation.

data This folder contains synthetic and real scenes with the setting files for them.